Ex. E to Cramer Declaration
PUBLIC REDACTED VERSION

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

IN RE GOOGLE PLAY STORE ANTITRUST LITIGATION

THIS DOCUMENT RELATES TO:

Epic Games, Inc. v. Google LLC et al., Case No. 3:20-cv-05671-JD

Case No. 3:21-md-02981-JD

MICHAEL D. ERNST DECLARATION IN SUPPORT OF EPIC'S RESPONSE TO GOOGLE PROFFER REGARDING EPIC'S PROPOSED INJUNCTION

ASSIGNMENT

1. I have been retained by Epic Games, Inc. ("Epic") in the matter captioned *Epic Games Inc. v. Google LLC et al.* I understand that pursuant to a jury verdict delivered on December 11, 2023, Epic proposed a Permanent Injunction on April 11, 2024, with certain remedies aimed at increasing competition in the Android App Distribution Market. I also understand that on June 24, 2024, Google submitted a Proffer along with supporting declarations from four Google employees estimating the technical work and costs associated with effectuating three of the remedies in Epic's Proposed Injunction.

2. Epic has asked me to evaluate whether Google's proposed changes are technically necessary and sufficient to effectuate the remedies. Epic has also asked me to evaluate whether other designs, besides the ones Google proposed, are possible, especially ones that are simpler, cheaper, or less obstructive. And Epic has asked me to evaluate the resourcing, timeline, and costing estimates in Google's proposal, as well as those of any simplified or alternative proposals.

3. Epic has informed me that Dr. James Mickens of Harvard University has been asked to review and evaluate the security arguments raised in Google's Proffer. I understand that Epic has also asked Dr. Mickens to evaluate whether my proposals create additional security concerns.

4. My opinions are based on my analysis of sworn testimony, emails, internal memos and documents, presentations, other documents produced in this case, and publicly available material, as well as my extensive research and experience in software development. If more information becomes available or I perform further analysis, I may modify or supplement my opinions.

5. The Court asked Google for "a Proffer stating in detail the tech work required and economic costs, if any, to provide 'Catalog Access' and 'Library Porting'" and optionally for "distribution of third-party app stores."[1] There are at least three problems with Google's Proffer and the supporting declarations: (1) they often stray beyond technical concerns; I have focused my statement primarily on the technical aspects; (2) though lengthy, they lack design details that are necessary to fully

[1] DKT 978.

evaluate Google's proposals; I have tried to fill in some of the missing detail; and (3) Google's cost estimates are too high and are unsupported. There are two standard ways to budget for a software project: either (1) list every component of the cost in detail (for example, as function points) and sum the (relatively small) costs of each or (2) use actual costs for a comparable project. [2] I have done the former. Google has done neither. Google says it does not know exactly what work is necessary,[3,4,5] but it also claims there is "zero buffer" for headcount and duration in its estimates (even though those numbers are ranges).[6] Google gives no concrete costs for similar or related previous projects. Google often claims the need to build something that already exists. Google's estimates for specific tasks are also too high, and in some cases they rely on the say-so of a particular declarant rather than on facts .[7]

6. I thank the staff at Keystone Strategy, LLC for their help. All opinions are my own.

7. I am paid $900 per hour for my time. My compensation does not depend upon the outcome of this matter or any opinion that I reach. Throughout my career, I have donated all my consulting income to charity, and I will do the same for this case.

QUALIFICATIONS

8. I am the Eggers Professor in the Paul G. Allen School of Computer Science & Engineering at the University of Washington (Seattle campus). My primary technical interests are in software engineering, programming languages, type theory, security, program analysis, bug prediction, testing, and verification. My research combines theory and experimentation, with an eye to changing the way that software developers work.

9. I was previously a tenured professor at MIT, and before that a researcher at Microsoft Research. I have also worked at Facebook (now Meta), Amazon, and other tech companies.

---

[2] *See* https://en.wikipedia.org/wiki/Function_point.

[3] Baccetti Dep. Tr. 98:21-23.

[4] Baccetti Dep. Tr. 118:10-16.

[5] Baccetti Dep. Tr. 173:16-174:9.

[6] Baccetti Dep. Tr. 376:7-17.

[7] Baccetti Dep. Tr. 287:18-20.

10. I am an ACM Fellow (2014) and IEEE Fellow (2021) and received the CRA-E Undergraduate Mentoring Award (2018), the inaugural John Backus Award (2009), and the NSF CAREER Award (2002). My research has received an ACM SIGSOFT Impact Paper Award (2013), three ISSTA Impact Award Awards (2018, 2019, 2024), an ICSE Most Influential Paper Award (2017), a FSE Most Influential Paper Award (2024), 9 ACM Distinguished Paper Awards (ICSE 2018, FSE 2014, ISSTA 2014, ESEC/FSE 2011, ISSTA 2009, ESEC/FSE 2007, ICSE 2007, ICSE 2004, ESEC/FSE 2003), an ECOOP 2011 Best Paper Award, honorable mention in the 2000 ACM doctoral dissertation competition, and other honors.

11. In 2013, Microsoft Academic Search ranked me #2 in the world, in software engineering research contributions over the past 10 years. In 2016, AMiner ranked me #3 among all software engineering researchers ever.

12. I have written over 300 technical papers. I have published 9 papers in HCI conferences, I lecture about user interfaces, and I evaluate (e.g., grade) students' user interfaces as part of their coursework. I have published 8 papers about Android mobile applications, and I regularly evaluate students' Android apps as part of their coursework. In addition to being a researcher, I am an active software developer. In cooperation with my colleagues, I have developed and released dozens of software packages, several of which are widely used. See Appendix A for my Curriculum Vitae along with a partial list of software packages. My web page is https://homes.cs.washington.edu/~mernst/.

OVERVIEW

13. Epic's Proposed Injunction aims to promote competition in the market for the distribution of Android apps ("Android App Distribution Market") via remedies including: (1) Play Store Catalog Access, (2) Ownership Transfer (which it calls "Library Porting"),[8] and (3) the Play Store Distributing Third-Party App Stores.[9] Below is a brief overview of these remedies, Google's position from its Proffer, and my response. More details appear later in this document.

[8] Epic's Proposed Injunction Pg. 8 Ln 1-3.

[9] Epic's Proposed Injunction Pg. 8 Ln 24-26.

14. Epic's "catalog access" remedy would allow users of third-party app stores to browse and/or search both the store's own apps and those in the Play catalog. The Play Store remains responsible for installation of Play Store apps that are not distributed by the third-party app store, just as if the user had browsed or searched within the Play Store itself.

15. Google proposes providing catalog access to third-party app stores via a daily catalog export. Google's proposal contemplates limiting the information Google would make available to third-party stores, setting eligibility criteria for stores that could receive access to the catalog, requiring developers to opt-in to have their apps reflected in the catalog exported to third-party stores, and charging third-party stores for the right to access the catalog. Google estimates the cost of implementing this remedy, including a 30% buffer, to be between $27.5 million and $65.9 million, and estimates it would take 12-16 months to complete.[10]

16. I agree that catalog access could be achieved by exporting the catalog that Google already maintains and updates for use by the Play Store itself. However, none of the limitations and requirements in Google's proposal are technically necessary, nor are they consistent with the purpose of Epic's Proposed Injunction. An alternative solution would be to give third-party app stores access to the catalog query APIs already used by the Play Store. I estimate that catalog access can be accomplished in 2-3 months at a cost of less than $605K, with an ongoing cost of less than $50K/year and 1 month per year.

17. Epic's "library porting" remedy permits a user to transfer ownership for updates of an app from one app store to another. I call this "transferring app ownership."

18. Google argues that it need not implement Epic's proposed remedy because Android 14's update ownership permission is close enough to the remedy requested by Epic. But Android 14 lacks two features that are key to Epic's proposed remedy: it has no mechanism for ownership *transfer* and it lacks a mechanism for a user to transfer ownership of *multiple* apps in one step. Google acknowledges it could implement these additional features. Google estimates the cost of this remedy

[10] Cramer Decl. ¶12; Baccetti Decl. ¶36.

to be approximately from $1.7 to $2.4 million, and estimates that it would take one year to implement.[11]

19. In my view, library porting can be solved by using or copying (1) an existing field in an existing data structure on the user's phone indicating app ownership and (2) an existing mechanism in the app manifest indicating which app stores are permitted to distribute the app. Android would consult these two data points when performing relevant actions. I estimate that implementing this change would take 1 month and cost less than $320K.

20. Epic's "distributing third-party app stores" remedy requires that Google accept app stores for distribution through the Play Store, allowing users to install third-party app stores from the Play Store just like any other app.

21. Google proposes that if it is required to distribute third-party stores on the Play Store, it would vet such third-party stores under a new set of review policies and would further vet every app distributed by any such third-party store, subjecting all apps in the third-party store's catalog to a full-blown app review for compliance with Google's content and other policies. To build, implement, and maintain this remedy, Google estimates a cost including a 30% buffer, to be between $31.4 million to $66.7 million, and estimates this work would take 12-16 months. Google also estimates that it would cost around [redacted] million annually to scale Play's app review process.[12] For two to six years, Google estimates this cost, including a 20% buffer, to be between [redacted] and [redacted] million.[13]

22. Google's design includes eligibility and review requirements that are unnecessary and are not mandated by Epic's Proposed Injunction. The Play Store can distribute third-party app stores *today* and would require only minor technical changes to comply with the parity requirements of Epic's Proposed Injunction. I estimate that implementing this change would take 1 month and cost less than $95K.

[11] Cramer Decl. ¶13.

[12] Kleidermacher Decl. ¶21.

[13] Google Proffer Pg. 2 Ln 23.

DEFINITIONS

23. Here are my definitions of important terms.

24. An **app store** is an application that enables the user to install other apps.[14] A typical Android user would agree with this commonsense definition, and Google sometimes uses it as well.[15] I do not make a distinction between an app store and a launcher. An app may have both app store functionality and other functionality.

25. Many of Google's stated concerns stem from its different interpretations of "app store" in different parts of their documents. Use of consistent definitions eliminates these concerns. Google's main definition of an "app store" is an app with the **INSTALL_PACKAGES** permission. Such an app can install or update apps on the phone at any time, without user approval. That definition is inconsistent with how the term app store is commonly understood and used. However, Google does not always use that definition of "app store."[16,17] In addition, Mr. Kleidermacher claims that at the "Android Operating System" level, "app store" has no definition at all.[18]

26. A **superuser app** is an app with the INSTALL_PACKAGES permission. A given app may be an app store, may be a superuser app, may be both, or may be neither. Today, pre-installed app stores, including the Play Store, are superuser apps. All non-preinstalled app stores are *not* superuser apps. They have different capabilities, as explained immediately below.

27. An app store that is not a superuser app uses the **REQUEST_INSTALL_PACKAGES** permission. An app store with that permission can request that Android install the app, after which Android pops up a confirmation dialog box to verify the user's intent. This Android-controlled confirmation is necessary because a malicious app store could request (from Android) installation of app A, even though the user had actually clicked a button in the app store requesting installation of app B or had not clicked any button at all. Because Android controls the confirmation dialog box,

---

[14] *See* https://en.wikipedia.org/wiki/App_store.

[15] Kleidermacher Decl. ¶6.

[16] Baccetti Decl. ¶24.

[17] Google Proffer Pg. 22 Ln 14-17.

[18] Kleidermacher Dep. Tr. 113:11-14.

Android is sure to be following the user's desires regardless of the behavior of the app store itself. Unfortunately, this adds one extra click of friction to non-superuser app stores.

28. An app store **distributes** certain apps.[19] Upon a user request, the app store facilitates the installation of[20] or **installs**[21] or downloads[22] certain apps. On a particular device, each app may be **owned** or update-owned by one app store, meaning that app store is the one that updates the app. In the context of a third-party app store, a **Play Store app** is an app that the Play Store distributes, but the third-party app store does not.

29. Under Epic's Proposed Injunction, some but not all app stores will receive access to the Play catalog. These app stores **list** (but do not necessarily distribute) apps in the Play catalog. Listing benefits the developers of a listed app and the Play Store itself.[23] Users of such a third-party app store can browse and search the Play catalog, but the Play Store remains responsible for app installation and updating.

30. Google uses the term "change to the **Android Operating System**" for any change to Android, even a dialog box.[24] In standard computer science terminology, a software installer like the Play Store is *not* a part of the Operating System ("OS").[25] When Google's documents say a change to the Android Operating System is needed, the documents just mean that some code change is needed, possibly a small change in a simple part of Android.

CATALOG ACCESS

31. Epic's Proposed Injunction requires that Google "allow Third-Party App Stores to access the Google Play Store's catalog of apps not then available on those Third-Party App Stores."[26] If a user of a third-party app store wishes to install a Play Store app that is listed but not distributed by the

---

[19] Google Proffer Pg. 2 Ln 2-3.
[20] Google Proffer Pg. 24 Ln 5.
[21] Google Proffer Pg. 24 Ln 28.
[22] Google Proffer Pg. 22 Ln 5.
[23] Similarly to how Alley Oop benefits developers and the Play Store.
[24] Google Proffer Pg. 15 Ln 12-17, Pg. 16 Ln 8-9.
[25] *See* https://en.wikipedia.org/wiki/Android_(operating_system).
[26] Epic's Proposed Injunction, Pg. 7 Ln. 6-8.

third-party store, the Play Store would install that app through a "process similar to the Alley Oop integration offered by Google to certain third-party Developers."[27] Google's proposal on catalog access is supported primarily by the declaration of Mr. Vitor Baccetti. Google uses the term "metadata" to refer to data about the apps in its catalog.

32. I agree that exporting the catalog to third-party app stores is one way to implement the "catalog access" remedy. However, Google proposes restrictions and limitations that are unnecessary and inconsistent with the Proposed Injunction. This section discusses Google's proposal and how it should be modified to achieve effective catalog access, consistent with the injunction. In particular, I discuss what data should be included in the catalog provided to third-party app stores, how it should be provided, how frequently the data should be updated, whether app developers should opt-in or opt-out of catalog access, and the costs Google would incur to implement and maintain the remedy.

A. What catalog data is shared.

33. Mr. Baccetti proposes that Google export certain data about the apps in the Play catalog into a database for third-party app stores to access.[28] Mr. Baccetti does not define exactly what data will be included. To provide equally effective catalog access on third-party stores as users get on the Play Store, Google ought to provide third-party app stores access to the same data about the apps in the Play catalog that it uses for the Play Store. Google could do so by providing access to the same dataset or to a copy of it.[29] App data about apps in the Play Store catalog, including overall user ratings, is widely available, including in Bing, a non-Google search engine.[30]

34. Mr. Baccetti proposes that Google would not share any user data such as reviews of the app given by other users or data generated by Google, like "auto-translations, age ratings and install counts."[31] There is no technical reason to withhold this information, and in other instances Google

---

[27] Epic's Proposed Injunction, Pg. 7 Ln 12-13.

[28] Baccetti Decl. ¶8.

[29] At the very least, the catalog data provided to the third-party app stores should contain what developers are required to provide to Google when submitting their app to Play for review. Developer app information list, *See* https://mattoakes.net/guides/what-information-is-needed-for-a-google-play-store-listing/.

[30] *See* https://www.bing.com/search?q=android+app+spotify.

[31] Google Proffer Pg. 6 Ln 12-17.

does provide analogous information. For example, in Google's Maps Platform under the Places API, Google provides user data to developers, including user ratings and reviews for establishments, prominent points of interest, and geographic locations.[32] Google's suggestion that there are security or privacy concerns with providing user reviews is belied by the fact that this information is publicly displayed in the Play App and publicly accessible and scrapeable through any web browser on any platform.[33] The same is true of Google-generated data, including auto-translations, age-ratings, and install counts. And as noted above, to provide equally effective catalog access on third-party stores as users get on the Play Store, Google ought to export the same data that it uses to enable browsing and searching the catalog on the Play Store.

B. Format of the catalog.

35. Google suggests two different mechanisms for giving third-party app stores access to the Play catalog: regularly *export* the existing database or permit third-party app stores to *query* the database.[34] Both approaches are simple and easy, despite Google's claims to the contrary.

36. Exporting the catalog would not be difficult for Google to accomplish. It is a small dataset of about 3 million rows. Google supports datasets that are much larger and can reuse that expertise to export the data.

37. Alternatively, Google could enable queries to satisfy the catalog access remedy. This would not be difficult for Google to accomplish. Google has already implemented a mechanism to allow queries of its catalog. The catalog is queried by the Play Store website, and it is queried every time someone does a search in the Play Store app. Google just needs to permit third-party app stores to do exactly the same queries. Given that the catalog already handles all the traffic of the Play website and app, it will have no problem with third-party app stores. (I believe that query traffic will not increase significantly, because many searches on a third-party app store would previously have been searched on the Play Store.)

---

[32] *See* https://developers.google.com/maps/documentation/places/web-service/details.

[33] *See* https://play.google.com/store/apps?device=phone.

[34] Baccetti Decl. ¶28.

38. Google's claims about the complexity of this process are categorically false. All that Google would need to do to enable queries by third-party stores is to allow third-party stores to search the existing databases using existing mechanisms or trivial variants of them. I would not expect this to require a high level of integration or increase the implementation time. Connections like this are easy to make. I also do not expect query traffic to increase significantly as a result of allowing third-party stores to use this database, because many searches on a third-party app store would previously have been searched on the Play Store.

39. Google's last objection to enabling queries by third-party stores is that it would "deprive third-party app stores of the ability to differentiate themselves." That is not the case. Third-party app stores can differentiate themselves with their own apps. Furthermore, a query function can be useful to third-party stores.

C. Freshness of the catalog.

40. Mr. Baccetti proposes that Google would "regularly export [catalog] data" and "refresh [catalog data] on a daily basis."[35] To ensure that catalog access on third-party stores is as effective as catalog access on the Play Store, Google should update the catalog for third-party app stores as often as Google updates the catalog that is used by the Play Store itself. Mr. Baccetti states that near real-time parity of catalog data information with the Play Store is technically feasible.[36]

41. If Google chooses to export the catalog, then there are two well-known approaches to keeping third-party app stores in sync with the Play catalog. One is a "pull" model: app stores can choose to query the catalog server as often as they want (e.g., every five minutes), and Google would not place any restrictions on third-party app stores' ability to query the catalog server. The other is a "push" model: Google can send an update to the third-party app stores each time the Play Store catalog is updated. Google could achieve a push model by simply sending add, modify, and delete commands to third-party app stores. Both "pull" and "push" models are standard approaches for syncing

[35] Baccetti Decl. ¶8.

[36] Baccetti Dep. Tr. 94:10-22.

information across multiple sources. Alternatively, if Google were to choose the query mechanism for catalog access, then third-party app stores would always have up-to-date information.

D. App Developer Consent for Catalog Access.

42. As part of implementing catalog access, Mr. Baccetti proposes that Google would "give developers a mechanism to provide consent to participate in catalog access through the Google Play Console… [and] provide [app] developers who opt in to catalog access with additional options that allow the developer to be more specific in selecting which authorized third-party stores they want Google to give access to their apps."[37] In other words, Google proposes that app developers would have to opt-in to have their apps listed in third-party stores, then further opt-in to which specific third-party app stores can list their app. In this mechanism, by default app developers would not have the benefit of being exposed to more potential users on third-party stores.

43. An opt-in mechanism adds no security, yet it has clear downsides: it would add complexity to the implementation, and it would reduce the effectiveness of the remedy. Google hypothesizes a malicious app store that intentionally distributes malware and illegal content,[38] but this is a strawman argument that ignores the tools Google has to address and remove malware on GMS devices, including Google Play Protect ("GPP").

44. Google speculates that developers may want to limit their apps to certain app stores.[39] Google gives no examples, only far-fetched hypotheticals. As a software developer, I would want my product to be available as widely as possible. Google hypothesizes that a software developer might not want their app distributed in the same store as some other app.[40] This is an unlikely concern for a developer that already distributes its app through the Play Store, which has a catalog of over 3 million apps.

---

[37] Baccetti Decl. ¶19.

[38] Google Proffer Pg. 11 Ln 13-15.

[39] Google Proffer Pg. 10 Ln 9-11.

[40] Baccetti Decl. ¶20.

45. If – unrelated to the requirements of the Proposed Injunction and contrary to spirit of the remedy – Google wishes to address these rare edge cases, an opt-out system is no more technically difficult and offers the same choices to developers, without reducing the effectiveness of the remedy.

E. Cost, resource, and timeline estimates

46. Google states that its proposal would require "establishing and maintaining server-to-server or client-to-client connections between Google and the third-party app stores."[41] Google's Proffer further states that this would also require Google "to build, support, and maintain servers[42] to handle the traffic of users browsing another app store."[43] Google then mentions that this would be "extremely challenging and costly", estimating the cost of "implementing [the catalog access] remedy to be approximately $13.6 million to $23.7 million" with "ongoing maintenance and policy enforcement cost of $7.5 million to $27 million, depending on the duration of the injunction."[44,45,46] Google also estimates that it would take "12-16 months to implement this remedy."[47] However, virtually every company that provides online services to mobile or desktop applications provides exactly what is described here – API endpoints that the application sends requests to, and in return receives results. This is not novel, challenging, time-consuming, or costly. Describing it as "deep[] technical integration"[48] without justification is absurd. Mr. Baccetti admits that access to this information is already happening in the form of "scraping" the publicly available information from the Play Store."[49] The reason this is already happening is that it is very simple to do.

---

[41] Google Proffer Pg. 6 Ln 22-23.

[42] Building, supporting, and maintaining a server is no burden to Google. Google doesn't distinguish between "supporting" and "maintaining", which sound like the same thing to me. Google has over 2.6 million servers (maybe more). *See* https://en.wikipedia.org/wiki/Google_data_centers. If the expected lifetime of a server is 3 years, then Google builds a server more than 5000 times per business day.

[43] Google Proffer Pg. 6 Ln 28-Pg. 7 Ln 29.

[44] Google Proffer Pg. 7 Ln. 6, Ln 20-23.

[45] Baccetti Decl. ¶¶32-35.

[46] Cramer Decl. ¶12.

[47] Baccetti Decl. ¶36.

[48] Baccetti Decl. ¶ 29.

[49] Baccetti Decl. ¶ 20.

47. Google's estimate includes technical work, such as an opt-in mechanism for app developers and developing a fee structure for third-party app stores, that is both unnecessary and contrary to the goals of the injunction.

48. Table 1 summarizes Mr. Baccetti's effort estimates for Catalog Access by resource role. Google estimates that the implementation of Catalog Access would require the attention of 38 full-time software engineers and ten managers (Technical and Product), one technical writer, and one designer for a period of up to 9 months.[50] In addition, Google estimates maintenance and support will require 6 engineers.[51]

[50] Baccetti Decl. ¶33.

[51] These estimates do not include non-technical staff like Legal, Business Development, Finance, and others as shown in Table 1.

Table 1: Google's effort estimates for Catalog Access[52]

Catalog Access	Google's Projected Headcount			
	3 Months	6-9 Months[53]	24-72 Months	Total
Technical Personnel				
SWE (Software Engineer)	1	37	4	42
PM (Product Manager)	1	4		5
TPM (Technical Program Managers)	1	4		5
TSC/TSE (Technical Solutions Engineer)	3			3
SRE (Software Reliability Engineer)			2	2
TW (Technical Writer)		1		1
UX Designer		1		1
Non-Technical support				
Legal Epic		3		3
Business Dev Managers		2		2
Finance		2		2
T&S (Trust and Safety)			2	2
Tax		2		2
XFN Resources, Program Manager		2		2

49. Google's estimates are far too high. For example, Google does not give actual current staffing costs for maintenance and support to compare to their estimate. Google does not explain why it needs 3 legal personnel devoted full-time for 6-9 months, doing nothing but "Implementation & Launch." Google does not explain why a full-time programmer is needed for 6-9 months in a policy task whose output is text that can easily be incorporated into existing consent mechanisms, especially when "sync & developer consent" has 8 programmers assigned to it. The "mechanism for more frequent updates" is not needed given current infrastructure, but even *if* the export mechanism were too slow, 1 or 2 subject-matter experts on a particular part of the process would likely spend a week or two to solve it, rather than 10 people full time for 6-9 months. The automated signals Google mentions are, to the best of my knowledge, already implemented, and if Google doesn't know which automated signals it

[52] In the final estimate, these numbers are inflated by a 30% buffer.

[53] 1 SWE resourced for 6 months.

will measure, it is not reasonable to claim that exactly two people will be needed for the duration of the project to implement them – especially if the claim that the estimates contain no buffer is taken at face value.

50. For my own assessment of costs, I break down catalog access into key technical components that must be designed, implemented, and maintained.

51. **Server:** Google already has the catalog data on hand stored in an accessible server. Google uses this data to populate the web interface for the Play Store and the Play Store app.[54] Google can re-use the server or adapt its design for third-party app stores.

52. **Alley Oop:** Mr. Baccetti proposes that apps listed by third-party app stores, but distributed by Play, would be installed via a process "similar to the Alley Oop" process.[55] Google could simply reuse Alley Oop or its successor. Alternatively, if small changes are needed, reuse of existing components should make this quick and easy, and reuse offers additional benefits such as consistency between systems and reduced maintenance costs. Any modifications to Alley Oop will be much easier than building all of Alley Oop from the ground up. As a point of comparison, in 2017 Google proposed building "Alley Oop turn around" [56,57] a "significant latency-oriented rewrite"[58] to address latency issues; the following staff were allocated to the project: 1 TL for 3 months, 1 Client UI Eng for 3 months, 1 Client Eng for 3 months, 1 Analytics Eng for 3 months, 1 Eng Prod for 1 month, 1 UX support for 1 month, 1 BA support for 1 month.[59]

53. Mr. Baccetti states that the user must be "signed in with a Google account" before proceeding with the app installation.[60] Mr. Baccetti proposes that if the app package does not exist or the user or

---

[54] *See* https://play.google.com/store/games?hl=en_US.

[55] Baccetti Decl. ¶17.

[56] Google says that this project was a significant latency-oriented rewrite to fix what was wrong with Alley-Oop.

[57] The Alley-Oop Turn around project involved 26 distinct tasks.

[58] GOOG-PLAY-001586531 at 31.

[59] GOOG-PLAY-001586531 at 36-37.

[60] Baccetti Decl. ¶12.

device is not permitted to install the app, "the user will see a message informing the user that the app is not available."[61] Google has already implemented all of these checks in the context of Alley Oop.[62]

54. Mr. Baccetti proposes that Google may need to build and implement an additional security layer that ensures that app stores calling the API have been authorized by the developer to distribute the app. Again, Google has already implemented this authentication check as part of Alley Oop.[63]

55. **Maintenance costs:** Once established, little maintenance of the hardware or software (beyond what is done on a large scale by centralized teams) is required to maintain the catalog access system described above.

56. Table 2 lists the technical components of Catalog Access. For each component, I describe existing functionality and infrastructure and my proposed changes. I provide resource and timeline estimates for the additional work needed to effectuate the remedy.

57. Google has best-in-class developer infrastructure: whole-codebase editing, build systems, compiler analyses, test automation, and more.[64,65] Google is noted for its attention to detail and code quality. My estimates assume that the Android code conforms to standard industry practices, such as that the code is documented, tested, modularized, and reusable. Google can build on existing functionality and features. In rare cases when code cannot be reused, developers can still reuse the design, logic, tests, and the knowledge gained while building and using it.

58. After the code is complete, some tasks are required before the code is in consumers' hands. Google's documents do not discuss these tasks. Google already has mechanisms for this process that it uses with every release (major and minor) and can re-use without change and without incurring any new costs. For example, Google distributes changes to OEMs, and Google copies the compiled code to a server that automatically deploys it to Pixel phones.

---

[61] Baccetti Decl. ¶13.

[62] Alley Oop install flow GOOG-PLAY-001040084 at 88-89.

[63] Alley Oop install flow GOOG-PLAY-001040084 at 88.

[64] "Lessons from building static analysis tools at Google", Sadowski et al, CACM 2018.

[65] Google enforces policies, procedures, code quality by enforcing code review for all changes to the code base and Google's automated release system produces a report of all changes contained in a release. *See* https://sre.google/sre-book/release-engineering/.

Table 2: My effort estimates for Catalog Access

Component	Existing functionality and infrastructure	Scope of new functionality and infrastructure	Est. resources	Est. timeline
Catalog data server	Google already has the catalog on hand stored in an accessible server.	For the "query" design: N/A For the "export" design: Reuse the existing server or create a copy of it.	1 SWE 1 PM	1 month
Catalog data APIs	These APIs already exist internally for Google's catalog data server. Google has extensive experience releasing SDKs and supporting documents.[66]	Expose APIs to eligible third-party stores. Documentation for third-party stores to integrate catalog access.	1 SWE 1 TW	1 month
Scale Alley Oop	Alley Oop is already scalable. As of 2020 Alley Oop was responsible for 50m+ installs a week, representing 6% of total Play Store installs.[67] AO UX has already been designed.[68]	N/A	N/A	N/A
App store registration	Alley Oop SDKs and APIs exist and have been rolled out to numerous partners (e.g. Facebook, AdMob, Pinterest, YouTube).	Management platform for third-party stores to register for Alley Oop. Mechanism to expose an authorization letter for app developers based on signing keys.	1 SWE 1 TPM 1 UX	2-3 months
Server and software maintenance			1 SRE	1month/year
User reviews and ratings (optional)	User reviews and ratings already exist in the dataset. Google exposes user reviews and ratings on (for example) the Google Maps Places API. Play catalog access could be similar.	For the "query" design: N/A For the "export" design: Include this in the catalog.	1 SWE	1 month

[66] *See* https://cloud.google.com/sdk.
[67] GOOG-PLAY-001264032 at 33.
[68] GOOG-PLAY-001264032 at 35.

59. My estimate for the alternative "query" design includes 5 to 6 person-months of SWE ($118-281K), 1 person-month of TW ($34-41K), 2 to 3 person-months of TPM ($79-141K), and 2 to 3 person-months UX ($79-141K), for a total implementation cost of $310-$604k. Ongoing costs consist of 1 person-month of SRE ($39-47K) per year.

60. The "export" design would add an additional 2 person-months of SWE ($79-94K) and 1 person-month for a PM ($30-36K), for a total of $458-781K, at Google's option.

TRANSFER OWNERSHIP (ALSO KNOWN AS "LIBRARY PORTING")

61. Epic's Proposed Injunction requires that users have the ability to transfer ownership for apps, such that a different app store manages updates for the app on the device going forward.[69] Epic's Proposed Injunction also requires Google to support "bulk app transfer": transferring the ownership of multiple apps with a single click.[70]

62. Google incorrectly claims that existing Android 14 capabilities are sufficient to satisfy the injunction.[71] But Google concedes that the cost and burden of what is actually required by Epic's injunction are minimal.

63. I explain below why existing Android 14 functionality does not satisfy the injunction. Then, I propose a simple, secure design, building from Google's proposal, that re-uses existing functionality in Android.

A. Android 14 Capabilities

64. In response to Epic's Proposed Injunction, Google states that "Android 14 already enables third-party app stores to request user permission to update apps installed by other app stores."[72] Google reasons that "Epic's proposed remedy is unnecessary because Android's controls over cross-store updates already allow most of what Epic refers to in its Proposed Injunction."[73] I disagree.

---

[69] *See* https://source.android.com/docs/setup/create/app-ownership.

[70] Epic's Proposed Injunction Pg. 7 Ln 19-22.

[71] Google Proffer Pg. 13 Ln 14 -15.

[72] Google Proffer Pg. 13 Ln 6-7.

[73] Google Proffer Pg. 13 Ln 14-15.

Android 14 does not satisfy this section of the remedy, because it has no mechanism for ownership transfer of one or multiple apps.

65. As I understand it, before Android 14, any superuser app could update any app—regardless of which store initially installed that app—and could do so without notifying the user. These "unauthorized cross-store updates…[are] called 'app clobbering'."[74] Android 14 partially addressed this issue by introducing two modalities for app ownership. By default, an app is owned by the store that installed it, in which case only that store may update it.[75,76] Android 14 prevented some clobbering problems by forbidding non-owning app stores from updating an app automatically without obtaining user consent.[77] Android 14 also introduced the ability to "clear ownership." If ownership is cleared, the app no longer has a specific owner, and "any app store on the device (including but not limited to the app store that requested the permission) can then update the app."[78,79] In other words, when ownership is cleared, any app store can clobber, not just superuser apps.

66. In Android 14, the developer can declare which third-party app store may update ownership. This is done by placing "update-ownership" information in the manifest file.[80] The developer may also provide a list of app stores that are not allowed to take ownership of the app under the deny-ownership tag in the app manifest.[81] This declaration in the app manifest is how Android confirms whether an app store that is not the app's owner can update the app.

---

[74] Google Proffer Pg. 13 Ln 21.

[75] There are ways to opt into and out of different behavior, such as "deny-ownership", which I am omitting for simplicity.

[76] Google Proffer Pg. 13 Ln 27.

[77] Cunningham Decl. ¶11.

[78] Google Proffer Pg. 14 Ln 6-7.

[79] I believe this design in insecure and can be annoying to users.

[80] *See* https://source.android.com/docs/setup/create/app-ownership.

[81] Id.

B. A design for transferring ownership

67. Epic's Proposed Injunction requires that ownership can be "transferred." This changes which app store is the owner and is allowed to update the app. Google recognizes that implementing this transfer of ownership, for a single app or for several apps at the same time, is technically feasible.[82]

68. Google criticizes "transfer ownership" on the grounds that an app store could become the owner of an "app that that the app store does not actually distribute",[83] in which case Google says the app would receive no more updates, resulting in user frustration and potential security risks.[84] This hypothetical problem is far-fetched, and it is easily prevented.

69. Google can restrict transfers of ownership to stores that the app developer has explicitly authorized. Technically, this would be nearly identical to the current "update-ownership" mechanism. Mr. Cunningham and the Google Proffer propose this solution, so it is perplexing that they repeatedly discuss the dangers of unrestricted ownership transfer.[85,86,87,88]

70. This proposed design handles various edge cases. If an app store goes defunct (or stops distributing updates) while owning an app, another app store that is authorized by the developer in the app manifest can request ownership of the app. If all stores in the manifest are defunct, the developer can release a new version of the app whose manifest lists an active store that distributes the app. That store could install the update.

---

[82] Cunningham Decl. ¶ 30; Cunningham Dep. Tr. 221:16-222:13.

[83] Google Proffer Pg.17 Ln 1-2.

[84] This is not true, since an app store that distributes the app could take ownership back.

[85] Cunningham Decl. ¶¶24-25.

[86] Cunningham Decl. ¶28.

[87] Cunningham Decl. ¶31.

[88] Google Proffer Pg. 15 Ln 5-6.

C. Bulk Ownership Transfer

71. Epic's Proposed Injunction states that users should be able to perform app-by-app and bulk ownership transfer.[89] Again, Google recognizes that this update to Android 14's current design is technically feasible.[90]

72. Google proposes to implement bulk transfer via multiple changes in the app manifest.[91] Google's Proffer proposes that it would create a mechanism for app developers to indicate if each app is bulk transferable and which "third-party app stores are permitted to obtain ownership over the app by means of the bulk transfer."[92] Google does not justify these complexities, and in my opinion they are unnecessary.

73. Instead, I propose that developers include, in the app's manifest, a list of all third-party stores that are permitted to obtain ownership over the app. There is no need to separately opt into single or bulk transfer.

74. Google remarks in its Proffer that bulk update is "unnecessary and would have serious negative consequences for the security of Android users",[93] and "these changes would significantly harm Android users."[94] I disagree.

75. Mr. Cunningham states that app-by-app ownership changes are superior to bulk ownership updates because app-by-app ownership changes are "in keeping with similar decisions that users are asked to make in the course of using their device" and because Google does not want "to overload the user with unrelated or hypothetical decisions, which do not immediately pertain to the action they are performing."[95] However, his belief is based on a mistaken understanding of the use case for bulk ownership. Mr. Cunningham conceded that he was thinking of a user who is attempting to update an

---

[89] Epic's Proposed Injunction Pg. 7 Ln 17-23.
[90] Cunningham Decl. ¶ 19; Cunningham Dep. Tr. 219-220.
[91] Cunningham Decl. ¶31.
[92] Google Proffer Pg. 16 Ln 14-15.
[93] Google Proffer Pg. 13 Ln 10-11.
[94] Google Proffer Pg. 15 Ln 5-6.
[95] Cunningham Decl. ¶17.

individual app.[96] He acknowledged that he had not considered a use case in which the user had decided to migrate ownership of multiple apps over to a new store. He agreed that in such a use case, app-by-app ownership changes could create greater friction.[97]

76. Google similarly speculates that users might get confused when doing bulk transfer and may not fully understand the consequences when presented with a "one-time User permission" UI to bulk update apps.[98] Google can prevent this by building a different UI for different use cases. A user seeking to update ownership for a single app can be presented with a UI facilitating ownership change for that app. A user seeking to migrate ownership for all apps—or a specific list of apps—could be presented with a different UI facilitating that change.

77. Google also suggests that users may have a reason to have different apps updated by different stores. If true, multi-app transfer does not undermine this user preference; users can continue to transfer permissions on an app-by-app basis.[99] Bulk update raises no concerns beyond those for single app ownership transfer – security, implementation, or otherwise. Bulk update is a convenience to users.

78. Finally, Google suggests that bulk ownership changes could harm users because the non-Play Store to which ownership is transferred may not actually distribute all of the apps being migrated, and this could result in some of the apps never being updated.[100] This is nothing new — it is exactly the same concern as Google raised for single app transfer. The same mechanism handles it: a list of distributing app stores in the app's manifest. Alternatively, Android could defer the ownership change for each individual app until the first time that the "receiving" app store successfully updates the app. If the receiving app store never successfully updates a particular app, then the "old" app store (e.g.,

[96] Cunningham Dep. Tr. 173-174.
[97] Cunningham Dep. Tr. 178:14-22.
[98] Google Proffer Pg. 16 Ln 5-7.
[99] Cunningham Decl. ¶ 20.
[100] Google Proffer Pg. 17 Ln 1-6.

Play) can continue pushing updates to the relevant app indefinitely. Mr. Cunningham conceded that this is technically feasible.[101]

D. Cost, resource, and timeline estimates

79. Google's Proffer states that these "changes to the Android Operating System would be very costly",[102] and would "require extensive developer previews, beta testing, feedback from users and OEMs, and final bug fixes"[103] and ultimately "impose significant costs and technical work on Google."[104] Google further asserts that "new features implemented into the operating system take time to build."[105] In my opinion, Google's argument is overblown because the data structures (e.g., the update ownership data field in the app manifest and the fields that record the current on-device value) already exist, and no complex UI is needed.[106]

80. Google states that, if it was "ordered to implement these changes off-cycle, the cost to Google would be far higher, as Google would have to initiate a separate round of user, developer, and OEM testing and feedback described above."[107] However, the needed changes are minor, and Google has frequently released features more significant than these off-cycle.[108]

81. The Android AOSP documentation notes that the AOSP architecture contains many layers: the Kernel (the operating system), Native Daemons and Libraries, Hardware Abstraction Layer, Android Runtime, System Services, Android Framework, Android API, System API, Device Manufacturer Apps, Privileged Apps, and Android Apps.[109] According to Mr. Cunningham, changes to the Android OS would primarily affect the System Services layer, Android API, Android

---

[101] Cunningham Dep. Tr 200:5-12.

[102] Google Proffer Pg. 18 Ln 11.

[103] Google Proffer Pg. 18 Ln 16-18.

[104] Google Proffer Pg. 13 Ln 12.

[105] Google Proffer Pg. 18 Ln 21.

[106] Or ones that are only trivially different.

[107] Cunningham Decl. ¶53.

[108] As just one example, in Google Play services v02.23 updated on 01/18/2023 (i.e., an off-cycle release) Google added new developer features for Google and third-party app developers to support Device Connectivity, Location & Context, Machine Learning & AI, Maps and System Management & Diagnostics related developer services in their apps see https://support.google.com/product-documentation/answer/11412553.

[109] *See* https://source.android.com/docs/core/architecture.

framework and Privileged Apps. Mr. Cunningham also notes that changes for transferring ownership would not involve any modifications to the lower layers of the AOSP architecture, including the Android Runtime, Hardware Abstraction Layer, Kernel, and Native Daemons and Libraries.[110] The complexity of changes within Android depends on the layer at which the changes need to be made—the higher the layer the easier it is to make a change, and the likelier such changes can be made off-cycle with minimal security and user impact. From my experience, changes made to the "System Services" layer and above are considered easier relative to changes below "System Services". Per Mr. Cunningham's admission, his transfer ownership design requires changes at or above the "System Services" layer.

82. Table 3 summarizes Mr. Cunningham's resourcing and timeline estimates for Library Porting. Mr. Cunningham states that Google would have to "design and implement" the "ownership change permission", "associated APIs", "user interfaces", and "OEM configuration for preloads." However, Google has existing materials that could be re-used for development efficiency. While Mr. Cunningham's resource and timeline estimates are not as unreasonable as Mr. Baccetti's estimates for catalog access. Nonetheless, Mr. Cunningham's design is unnecessarily complicated, making his estimates still an overestimation of the work required to address Epic's remedy.

Table 3: Google's effort estimates for Library Porting

Library Porting	Google Projected Headcount				
	3 Months or Less	6 Months	12 Months	2 Months for (2-6 Years)	Total
SWE (Software Engineer)	1		2	1	4
PM (Product Manager)		1			1
Developer Relations Engineer	1				1
Technical Solutions Consultant	1				1
UX Designer	1				1
UX Researcher	1				1
Total	9				

83. To estimate needed resources for my proposal, I break down the Library Porting remedy into key technical components.

[110] Cunningham Dep. Tr. 100:5-19.

84. **Data structures and single app ownership transfer:** Android must set the owner field `setUpdateOwner`.[111] The Google Proffer says that to "implement Epic's 'change ownership' proposal, Google would [need to] create a capability in the Android Operating System to perform the update owner switch."[112] That field is already modified by the "clear ownership" code, so the capability already exists. Mr. Cunningham characterizes setting this existing field as "a new code path in the operating system."[113]

85. Android must check, before setting the field, that the targeted app store is one of the listed ones. This is a matter of checking membership in a list.

86. Android must permit apps to list multiple app stores, rather than only one, in the app's manifest. This is already syntactically possible. If it is not recognized by Android, then Google needs to make a copy of the existing routine that returns the owner, `getUpdateOwnerPackageName()`,[114,115] and then change the copy marginally so that it returns a list rather than a single item.

87. **Bulk transfer:** This requires a "for" loop. In computer programming, a "for" loop's body expresses an action once, but at run time more than one action occurs. It is a basic feature taught in each introductory programming class.

88. **UI:** Two lists must be displayed to the user. These are simple to design and implement. A link must also be made from the "Store" section of the "App info" screen.

89. Table 4 lists my effort estimates for Library Porting.

---

[111] *See* https://cs.android.com/android/platform/superproject/main/+/main:frameworks/base/services/core/java/com/android/server/pm/pkg/mutate/PackageStateWrite.java;drc=ec1ab3cae003c16f3c2e6789498b9f4ad34ba4bc;l=65.

[112] Google Proffer Pg. 16 Ln 23-26.

[113] Cunningham Decl. ¶30.

[114] *See* https://cs.android.com/android/platform/superproject/main/+/main:frameworks/base/core/java/android/content/pm/InstallSourceInfo.java;drc=47cecc3ce03dd7981a3c681f1b22c86df0786527;l=156.

[115] *See* https://developer.android.com/reference/android/content/pm/InstallSourceInfo#getUpdateOwnerPackageName().

Table 4: My effort estimates for Ownership Transfer (Library Porting)

Component	Existing functionality and infrastructure	Scope of new functionality and infrastructure	Est. resources	Est. timeline
Ownership data fields	Google has already created the data field for current app ownership and a field to list apps that are not permitted to obtain ownership.	Permit an app manifest to include more than one owner.	1 SWE	1 month
Ownership transfer	Google already has code to read the manifest and set the field.	Look up a list in the manifest, check the list, and set a field.	1 SWE	1 month
Bulk Transfer		A for-loop around single-app ownership transfer.	1 SWE	1 month
UI	Google has design guidelines for creating list UIs in Android.	Create lists (e.g., checkboxes) for single and bulk transfer ownership. Add a button to the "Store" section of the "App info" screen.	1 UX Designer 1 SWE	1 month
Documentation	Google has existing documentation.	Add brief documentation about this small change	1 TW	1 month
Support	Google has a developer relations program.	Address questions and concerns about this small change; minor added load, given adequate documentation	1 Developer Relations Engineer	1 month
Developer opt-out (optional)	Google has a management console for app developers.	Add a list (e.g., checkboxes) of app stores that the developer can select. Store it in the catalog or elsewhere. When providing catalog access, remove appropriate items.	1 UX Designer 2 SWE	1 month

90. My estimate includes 4 person-months of SWEs ($157-188K), 1 person-month of UX ($36-47K), 1 person-month of TW ($34-41K), and 1 person-month of developer relations engineer ($34-41K), for a total of $261-317K.

91. Developer opt-out would add another 2 person-months of SWEs ($79-94K) and 1 person-month of UX ($36-47K), for a total of $376-458K, at Google's option.

DISTRIBUTION OF THIRD-PARTY APP STORES

92. Epic's Proposed Injunction requires Google, for 6 years, to "allow distribution of competing Third-Party App Stores on the Google Play Store."[116]

[116] Epic's Proposed Injunction Pg. 7 Ln 24-26.

93. Google's Proffer states, "Google's implementation of this proposed remedy would involve four steps: (1) redesign the Play Store to accommodate the distribution of app stores; (2) implement a thorough ongoing vetting process for the policies, conduct and catalogs of app stores that request to be distributed through the Play Store; (3) change the Android Operating System; and (4) build and implement a charging model for third-party app store distribution."[117] Google estimates that implementing the distribution of third-party stores would cost up to $25 million, with up to $42 million in annual ongoing maintenance costs and another [redacted] million per year to conduct app review (all numbers include the 30% buffer). Because Google insists that implementing this remedy includes operating system changes, Google estimates it will take 12-16 months (without the 30% buffer) to implement.

94. Google's proposal is overly complicated and costly. It adds a number of processes and limitations that are not required by Epic's remedy and that Google does not currently use to address purported risks of third-party stores. Google can satisfy the remedy quickly and simply by distributing third-party app stores on the Play Store.

A. Redesign of the Play Store

95. Google's Proffer states that the Play Store "is designed to distribute apps, not app stores" and distributing app stores would "require a fundamental redesign of the Play Store." Mr. Kleidermacher conceded at deposition that the Proffer is incorrect on both points: given that every app store is an app, "the physical ability to install [app stores] is there."[118] Furthermore, for third-party app stores that already exist, there are no new security concerns to distributing them via the Play Store. [119]

96. The supposed "fundamental redesign" envisaged by Google would require minimal resources to implement. For example, Mr. Baccetti agreed that the resources required to create a space within the Google Developer console for a developer to declare their app as an app store would be "de

[117] Google Proffer Pg. 19 Ln 24-28.

[118] Kleidermacher Dep. Tr: 44:12-21.

[119] *See* https://www.businessofapps.com/guide/app-stores-list/.

minimis."[120] Similarly, designing a means for app stores to be identified as such to users within the Play Store would require adding one extra category to the current list of 49 categories.[121] Google has added new categories multiple times in the past, such as for games, dating, and dining.[122] Developers are already responsible for listing the category and tags for their apps in the developer console.[123]

97. Mr. Baccetti states the need to implement ways to allow developers to "agree to abide by Play Store policies",[124] but Google already has an infrastructure to prompt developers to agree to its policies within the Google Developer Console.[125] Mr. Baccetti states the need to "accept additional terms of service"[126], but Google's terms of service already have a section for apps that can install other apps.[127] Google can continue to use parts of this section, only removing the restrictions on the app's "core functionality" and "permitted functionality."

98. Mr. Baccetti also believes that Google should issue a "warning that advises users when they are about to install an app store."[128] This is not justified by any security concerns. Mr. Baccetti does not offer any technical reason for this additional warning. Furthermore, the user will have just clicked to install an app store. Google offers no evidence that users need this further handholding.

99. In other words, all the mechanisms needed to make app stores available on the Play Store either already exist or can be implemented with minor changes to existing Play Store and Google Developer Console functionality.

B. Vetting Process and Security

100. The Proffer states, "[i]f Google were ordered to distribute third-party app stores through the Play Store, then Google would subject the catalogs of those third-party app stores to the

---

[120] Baccetti Dep. Tr. 323-324.

[121] *See* https://support.google.com/googleplay/android-developer/answer/9859673.

[122] *See* https://android-developers.googleblog.com/2016/07/introducing-new-app-categories-from-art.html.

[123] *See* https://support.google.com/googleplay/android-developer/answer/9859673.

[124] Baccetti Decl. ¶ 38.

[125] *See* https://play.google/developer-content-policy/.

[126] Baccetti Decl. ¶ 38.

[127] Exhibit A Kleidermacher Decl. Pg. 24.

[128] Baccetti Decl. ¶ 39.

same rigorous review" for conformance to the Play Store's content policies.[129] In other words, Google would treat every app on every third-party app store as if the app had been submitted to the Play Store. This is not called for by the Proposed Injunction, nor is it technically necessary. Google does none of that today, for any non-Play app store.[130] Google has never systematically vetted the apps on pre-installed OEM app stores prior to distribution nor has Mr. Kleidermacher ever recommended that Google do so.[131] Mr. Kleidermacher says he believes the proposed vetting is needed to protect Play's reputation, and therefore he believes Google would "likely" adopt these measures, although Google apparently has no such plans and he did not discuss this proposal (or any of his proposals) with decision-makers at Google.[132,133]

101. For clarity, Google has the right to review the APK for every app that is distributed from Play. This includes the APK for each third-party app store. This is consistent with my view that Google should treat app stores the same as any other app in their catalog. However, Google should not review all the apps distributed by every app store.

102. Android's Google Play Protect ("GPP") system already scans all installed apps on GMS phones at the time of the installation.[134] If the app has never been scanned before, GPP performs a real-time code scan. GPP can already disable or remove malicious apps as well as display a pop-up message to the user regardless of where the app came from.[135] Furthermore, Android 15 allows Android to quarantine malicious apps.[136]

103. Google's reputational concern is non-technical, non-security-based, and hypothetical. Google justifies giving itself veto power over every app in every app store that is distributed on Play because otherwise, "the reputation for safety, security, and content moderation that the Play Store has

---

[129] Google Proffer Pg. 21 Ln 1-2.
[130] Kleidermacher Decl. ¶¶15-21.
[131] Kleidermacher Dep. Tr. 209-210.
[132] Kleidermacher Decl. ¶7.
[133] Kleidermacher Dep. Tr.167:4-12.
[134] *See* https://developers.google.com/android/play-protect/client-protections#scanning-services.
[135] *See* https://support.google.com/android/answer/2812853.
[136] *See* https://www.geeksforgeeks.org/android-15-introduces-quarantine-mode-to-protect-users-from-dangerous-apps/.

spent over a decade and billions of dollars building would be irreparably damaged."[137] This concern is diminished given that in 2023 the Play Store installed malware 600 million times.[138] In one incident in May 2024 alone, the Play Store was discovered to be serving 90 apps with malware accounting for 5.5 million installs.[139] Google seems to believe that if the Play Store distributes app store A, and app store A distributes malware, then customers will blame the Play Store specifically rather than the developer who released the malicious app or app store A. Google offers no data to substantiate this hypothetical. In any event, there is no technical need for the vetting Google describes and the costs associated with it.

C. Changes to the Android Operating System

104. Google argues that distributing third-party stores on the Play Store would be costly and time-consuming because it would require making changes to the Android Operating System. I disagree.

105. Many of Google's stated concerns stem from its unwarranted assumption that Epic's injunction somehow requires third-party app stores to be superuser apps. Epic's Proposed Injunction contains no such requirement. Epic's Proposed Injunction requires parity, in terms of friction such as number of clicks, between installs from the Play Store and third-party app stores. Google incorrectly reasons that the only way to achieve parity is to make all third-party app stores be superuser apps, which would create security vulnerabilities that would require costly workarounds to mitigate – and arguably could not be fully eliminated.

106. There is a simpler way to achieve parity in the short term: the Play Store should pop up a confirmation dialog seeking user consent after the user clicks to install an app, identical to the consent dialog for app stores with REQUEST_INSTALL_PACKAGES permissions. This would be quick, easy, and cheap to implement. The two-step install flow, if implemented, would still be simpler

---

[137] Google Proffer Pg. 21 Ln 25-27.

[138] *See* https://usa.kaspersky.com/blog/malware-in-google-play-2023/29356/.

[139] *See* https://www.bleepingcomputer.com/news/security/over-90-malicious-android-apps-with-55m-installs-found-on-google-play/.

than the iOS install flow is today. In order to download a new app on iOS, a user must click "Get" button and then authenticate via Face ID, or Touch ID, or password.[140,141]

107. After deployment of the remedy as noted above, Google could (if it wishes) work to eliminate the extra click in *all* app stores. There are many possible ways to do so. One way, as contemplated by Google, is a trusted Android UI component for installing an app.[142] The trusted UI component would act like the popup invoked for REQUEST_INSTALL_PACKAGES and would be under the control of Android. Since Android knows the text and context of the trusted UI component (for example, text that states the name of the app and a button that says "Install"), a click on it assures Android that the user wishes to proceed with the installation. This UI component could be embedded in a screen of an app store, similarly to how a frame can be embedded in HTML. Only an app with the REQUEST_INSTALL_PACKAGES permission may embed the trusted UI component. Other solutions may exist, but this design shows that it is technically possible to have a one-click install process without giving any app store the dangerous INSTALL_PACKAGES permission.

D. Cost, resource, and timeline estimates

108. The vast majority of the costs that Google points to are a result of their proposal to vet every app distributed by every third-party app store on the Play Store – a design choice Google proposes that is not required by the injunction.

109. By contrast, my design implements the requirements of the injunction at low cost.

110. Even if Google's heavyweight solution were adopted, Google's claimed costs are inflated. The claim that there would be a "20 percent increase in Play's app and update review workload" is unsupported by evidence.[143] In addition, Mr. Cramer made similar unsupported assumptions regarding costs to those for catalog access and library porting.[144]

---

[140] While users can remove the need to authenticate on free apps in their iOS settings, downloading a free app still requires the user to click an "Install" button after the "Get" button. *See* https://support.apple.com/en-us/119848.

[141] If the has user has downloaded the app before (e.g., re-downloading a deleted app), Apple requires only one click. See https://support.apple.com/guide/iphone/get-apps-iphc90580097/ios.

[142] Cunningham Decl. ¶61.

[143] Google Proffer Pg. 25 Ln. 22.

[144] *See* Cramer Dep. Tr. 93-94.

111. According to the staffing resource breakdown given in Mr. Baccetti's declaration, the below table shows the breakdown of app store distribution costs by role. Under the proposal set out by Google, the entire undertaking would range between 708 and 1,276 person-months among the different roles listed.

Table 5: Google's effort estimates for app store distribution[145]

3rd Party App Store Distribution	Projected Headcount			
	3 Months or Less	6-12 Months[146]	24-72 Months	Total
Technical Personnel				
SWE (Software Engineer)	2	19	5	26
PM (Product Manager)	2	4	2	8
PgM (Program Manager)		4	1	5
TPM (Technical Program Manager)	1	2		3
TSC/TSE		3		3
UX Designer	1	2		3
UX Researcher	1	1		2
Non-Technical support				
T&S (Trust and Safety)		8	1	9
TVC (Temporary Vendor Contractor)		4	1	5
Legal Counsel	1	3		4
Policy FTE	1	1	1	3
Ops FTE		1	1	2
XFN Government Affairs and Public Policy		1		1

112. By contrast, here is my estimate. Note that my estimate is for a different, simpler design than Google's.

[145] I believe Google can develop new UI features rapidly. For example, Google recently introduced Jetpack Compose and Material UI—UI tools in modern Android development designed to streamline the creation of intuitive user interfaces. Jetpack Compose, a modern UI toolkit introduced by Google, leverages a declarative approach, allowing developers to build UIs with less code and more expressive syntax. This approach significantly accelerates development cycles. Material UI, which provides a comprehensive set of design guidelines and components, ensures consistency and quality in app interfaces. The Play Store itself already uses Compose to some degree (Episode 185: Play Store *See* https://adbackstage.libsyn.com/episode-185-play-store) of the Android Developers Backstage podcast). The System UI team also already talked briefly about adopting Compose in the future in Episode 187: System UI: A Retrospective (*See* https://adbackstage.libsyn.com/episode-187-system-ui-a-retrospective).

[146] Includes role counts from strict 6-month stints, strict 9-month stints, and a 12-month stint.

Table 6: My effort estimates for third-party app distribution via Play Store

Component	Existing functionality and infrastructure	Scope of new functionality and infrastructure	Est. resources	Est. timeline
Redesign of the Play Store Interface	49 categories exist. Google should have admin tools or reusable code to add a new category to the Play Store.	Add 1 category to the current 49 categories, as has been done before.	1 SWE	1 month
Changes to operating system	None required	N/A		
Click parity of Play and other app stores	All required permissions already exist.	Either remove the INSTALL_PACKAGES permission from the Play Store or pop up an extra dialog from the Play Store.	1 SWE	1 month

113. My estimate includes 2 person-months of SWE ($79-94K) for a total of $79-94K.[147]

CONCLUSION

114. The Court requested an estimate of "the tech work required and economic costs" of implementing "distribution of third-party app stores", "library porting" (which I call "transfer of ownership"), and "catalog access."[148]

115. Google provided the outline of a complicated design that conflicts with the goals of the Proposed Injunction and suffers security flaws. Google estimates that implementing its design would take 12-16 months. Google estimates the cost as $61-137 million, plus up to [redacted] million for continuing support over the 6 years of the injunction. Google's estimates are too high even for the design Google proposes.

116. I provided a simpler and more secure design that fits the spirit of the Proposed Injunction. I estimate that implementing my design would take 3 months, and it would not need to wait for the next major release of Android. I estimate the cost as $0.8 million to $1.2 million, plus up to $0.3M for continuing support over the 6 years of the injunction.

[147] I use Mr. Cramer's rate cards for Google employee costs.

[148] DKT 978.

Pursuant to 28 U.S.C. § 1746, I declare under penalty of perjury that the foregoing is true and correct and that I executed this declaration on this July 24, 2024, in Seattle, WA.

Michael Ernst
Michael Ernst (Jul 24, 2024 19:58 PDT)

MICHAEL D. ERNST

Appendix A

MICHAEL ERNST

Curriculum Vitae

Computer Science & Engineering
University of Washington
Allen Center 538
Box 352350
Seattle, WA 98195-2350

Phone: 206-221-0965
Fax: 206-616-3804
Email: mernst@uw.edu
Web: https://homes.cs.washington.edu/ mernst/

EDUCATIONAL HISTORY

University of Washington, Seattle
Ph.D., Computer Science & Engineering
August 2000
Dissertation: Dynamically Discovering Likely Program Invariants

University of Washington, Seattle
M.S., Computer Science & Engineering
March 1997

Massachusetts Institute of Technology, Cambridge, MA
S.M., Electrical Engineering and Computer Science
September 1992

Massachusetts Institute of Technology, Cambridge, MA
S.B., Electrical Engineering and Computer Science
June 1989

EMPLOYMENT HISTORY

University of Washington
Seattle, WA
Professor, Sep. 2014 – present
Associate Professor, Jan. 2009 – Sep. 2014

MIT
Cambridge, MA
Associate Professor, July 2007 – Dec. 2008
Associate Professor (without tenure), Feb. 2005 – June 2007
Assistant Professor, Sep. 2000 – Feb. 2005

University of Washington
Seattle, WA
Research Assistant, Sept. 1996 – Aug. 2000

Rice University
Houston, TX
Lecturer, Sept. 1995 – May 1996

Microsoft
Redmond, WA
Researcher, Mar. 1993 – Aug. 1995
Software Design Engineer, Sep. 1992 – Mar. 1993

MIT
Cambridge, MA
Research Assistant, Sep. 1989 – Aug. 1992
Teaching Assistant, Jan. 1989 – June 1989

Texas Instruments
Dallas, TX
Summer intern, summers May 1986 – Sep. 1989

Technical consulting:
Facebook, June 2018 – September 2018
Samsung, Dec. 2009 – Dec. 2010
McKinsey & Co., Sep. 2007 – Jan. 2008
Kestrel Technology, Nov. 2006 – Dec. 2008
Institute for Defense Analyses, Jan. 2006 – Dec. 2008
Mercury Interactive, July 2004 – July 2005

Legal consulting:
Kelley Goldfarb Huck Rath & Riojas PLLC, November 2017 – November 2018
Sullivan Law Group, January 2016 – June 2016
VendNovation, August 2011 – July 2012
Sugarman & Sugarman, May 2008 – July 2009
GraniteStream, Oct. 2001 – May 2003

Awards and Honors

FSE Most Influential Paper Award, 2024
ISSTA Impact Award, 2024
Susan Eggers Endowed Professorship in Computer Science & Engineering, 2022
ECOOP Distinguished Artifact Award, 2022
IEEE Fellow, 2021
ACM SIGSOFT Outstanding Research Award, 2020
ISSTA Impact Award, 2019
ACM SIGSOFT Distinguished Paper Award, 2018
CRA-E Undergraduate Research Faculty Mentoring Award, 2018
ISSTA Impact Award, 2018
JavaOne 2017 Rock Star, September 2017
ICSE 2017 Most Influential Paper award, for ICSE 2007 paper, May 2017
Ranked 3rd among all software engineering researchers, AMiner, 2016
JavaOne 2016 Rock Star, September 2016
ASE 2015 paper nominated for Distinguished Paper Award, November 2015
Fellow of the ACM, 2014
FSE 2014 ACM Distinguished Paper Award, November 2014

ISSTA 2014 ACM Distinguished Paper Award, July 2014
Ranked 2nd among software engineering researchers worldwide, for work in the last 10 years, Microsoft Academic Search, 2013
UIST Best Paper honorable mention, 2013
ACM SIGSOFT Impact Paper Award, 2013
ESEC/FSE 2011 ACM Distinguished Paper Award, September 2011
ECOOP 2011 Best Paper Award, July 2011
Inaugural IBM John Backus Award, August 2009
ISSTA 2009 ACM Distinguished Paper Award, July 2009
JavaOne 2009 Rock Star, May 2009
ISSTA 2008 paper selected for expedited journal publication, July 2008
ASE 2007 paper selected for expedited journal publication, November 2007
ESEC/FSE 2007 ACM Distinguished Paper Award, September 2007
Ranked 5th among software engineering researchers worldwide in a CACM article, June 2007
ICSE 2007 ACM Distinguished Paper Award, May 2007
Most Innovative JSR of the Year (Sun Microsystems JCP Program), May 2007
ASE 2006 paper selected for expedited journal publication, September 2006
ISSTA 2006 paper selected for expedited journal publication, July 2006
IBM faculty award, June 2006
ASE 2005 paper nominated for Best Paper Award, November 2005
ICSE 2004 paper nominated for ACM Distinguished Paper Award, May 2004
Ross Career Development Professorship of Software Technology (MIT), July 2003
ESEC/FSE 2003 ACM Distinguished Paper Award, September 2003
IBM Eclipse Innovation Award 2003, 2004, 2005
VMCAI 2003 paper selected for expedited journal publication, January 2003
NSF CAREER Award, February 2002
ICSM 2001 Best dissertation of past three years, November 2001
Honorable mention, ACM doctoral dissertation competition, 2000
U. of Washington William Chan Memorial Dissertation Award, 2000
ICSE 2000 paper selected for expedited journal publication, June 2000
ICSE 1999 paper selected for expedited journal publication, May 1999

(The list does not include graduate fellowships or other graduate school and earlier awards.)

AFFILIATIONS AND OTHER APPOINTMENTS

Adjunct Associate Professor, Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA

PUBLICATIONS

This section avoids duplications by omitting previous conference, workshop, or TR publications that are superseded by a subsequent publication. It also omits tool demos and other ancillary publications, even when fully reviewed. A full list, with indications of relationships among papers, appears at `http://homes.cs.washington.edu/~mernst/pubs/`. Also note that in computer science, conference articles are at least as prestigious as journal articles.

Refereed archival journal publications

[1] Colin S. Gordon, Michael D. Ernst, Dan Grossman, and Matthew Parkinson. Verifying invariants of lock-free data structures with rely-guarantee and refinement types. *ACM Transactions on Programming Languages and Systems*, 39(3):11:1–11:54, May 2017.

[2] Ivan Beschastnikh, Patty Wang, Yuriy Brun, and Michael D. Ernst. Debugging distributed systems: Challenges and options for validation and debugging. *Communications of the ACM*, 59(8):32–37, August 2016.

[3] Ivan Beschastnikh, Patty Wang, Yuriy Brun, and Michael D. Ernst. Debugging distributed systems: Challenges and options for validation and debugging. *ACM Queue*, 14(2):91–110, March/April 2016.

[4] Kıvanç Muşlu, Yuriy Brun, Michael D. Ernst, and David Notkin. Reducing feedback delay of software development tools via continuous analysis. *IEEE Transactions on Software Engineering*, 41(8):745–763, August 2015.

[5] Ivan Beschastnikh, Yuriy Brun, Jenny Abrahamson, Michael D. Ernst, and Arvind Krishnamurthy. Using declarative specification to improve the understanding, extensibility, and comparison of model-inference algorithms. *IEEE Transactions on Software Engineering*, 41(4):408–428, April 2015.

[6] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Early detection of collaboration conflicts and risks. *IEEE Transactions on Software Engineering*, 39(10):1358–1375, October 2013.

[7] Adam Kieżun, Vijay Ganesh, Shay Artzi, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. HAMPI: A solver for word equations over strings, regular expressions, and context-free grammars. *ACM Transactions on Software Engineering and Methodology*, 21(4):25:1–25:28, November 2012.

[8] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. The HaLoop approach to large-scale iterative data analysis. *The VLDB Journal*, 21(2):169–190, 2012.

[9] Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst, Arvind Krishnamurthy, and Thomas E. Anderson. Mining temporal invariants from partially ordered logs. *SIGOPS Operating Systems Review*, 45(3):39–46, December 2011.

[10] Frank Tip, Robert M. Fuhrer, Adam Kieżun, Michael D. Ernst, Ittai Balaban, and Bjorn De Sutter. Refactoring using type constraints. *ACM Transactions on Programming Languages and Systems*, 33(3):9:1–9:47, May 2011.

[11] Shay Artzi, Adam Kieżun, Julian Dolby, Frank Tip, Danny Dig, Amit Paradkar, and Michael D. Ernst. Finding bugs in web applications using dynamic test generation and explicit state model checking. *IEEE Transactions on Software Engineering*, 36(4):474–494, July/August 2010.

[12] Shay Artzi, Jaime Quinonez, Adam Kieżun, and Michael D. Ernst. Parameter reference immutability: Formal definition, inference tool, and comparison. *Automated Software Engineering*, 16(1):145–192, March 2009.

[13] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1–3):35–45, December 2007.

[14] Lilian Burdy, Yoonsik Cheon, David Cok, Michael D. Ernst, Joe Kiniry, Gary T. Leavens, K. Rustan M. Leino, and Erik Poll. An overview of JML tools and applications. *Software Tools for Technology Transfer*, 7(3):212–232, June 2005.

[15] Toh Ne Win, Michael D. Ernst, Stephen J. Garland, Dilsun Kırlı, and Nancy Lynch. Using simulated execution in verifying distributed algorithms. *Software Tools for Technology Transfer*, 6(1):67–76, July 2004.

[16] Michael D. Ernst, Greg J. Badros, and David Notkin. An empirical analysis of C preprocessor use. *IEEE Transactions on Software Engineering*, 28(12):1146–1170, December 2002.

[17] Elizabeth L. Wilmer and Michael D. Ernst. Graphs induced by Gray codes. *Discrete Mathematics*, 257:585–598, November 28, 2002.

[18] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, February 2001.

[19] Michael D. Ernst. Playing Konane mathematically: A combinatorial game-theoretic analysis. *UMAP Journal*, 16(2):95–121, Spring 1995.

Conference proceedings and other non-journal articles – Fully refereed publications

[20] Martin Kellogg, Narges Shadab, Manu Sridharan, and Michael D. Ernst. Accumulation analysis. In *ECOOP 2022 — Object-Oriented Programming, 33rd European Conference*, pages 10:1–10:31, Berlin, Germany, June 2022.

[21] Zhen Zhang, Yu Feng, Michael D. Ernst, Sebastian Porst, and Isil Dillig. Checking conformance of applications against GUI policies. In *ESEC/FSE 2021: The ACM 29th joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 95–106, Athens, Greece, August 2021.

[22] Martin Kellogg, Narges Shadab, Manu Sridharan, and Michael D. Ernst. Lightweight and modular resource leak verification. In *ESEC/FSE 2021: The ACM 29th joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 181–192, Athens, Greece, August 2021.

[23] Rashmi Mudduluru, Jason Waataja, Suzanne Millstein, and Michael D. Ernst. Verifying determinism in sequential programs. In *ICSE 2021, Proceedings of the 43rd International Conference on Software Engineering*, pages 37–49, Madrid, Spain, May 2021.

[24] Martin Kellogg, Martin Schäf, Serdar Tasiran, and Michael D. Ernst. Continuous compliance. In *ASE 2020: Proceedings of the 35th Annual International Conference on Automated Software Engineering*, pages 511–523, Melbourne, Australia, September 2020.

[25] Wing Lam, August Shi, Reed Oei, Sai Zhang, Michael D. Ernst, and Tao Xie. Dependent-test-aware regression testing techniques. In *ISSTA 2020, Proceedings of the 2020 International Symposium on Software Testing and Analysis*, pages 298–311, Los Angeles, CA, USA, July 2020.

[26] Annie Louis, Santanu Kumar Dash, Earl T. Barr, Michael D. Ernst, and Charles Sutton. Where should i comment my code? a dataset and model for predicting locations that need comments. In *ICSE NIER, Proceedings of the 42nd International Conference on Software Engineering, New Ideas and Emerging Results Track*, pages 21–24, Seoul, Korea, May 2020.

[27] Martin Kellogg, Manli Ran, Manu Sridharan, Martin Schäf, and Michael D. Ernst. Verifying object construction. In *ICSE 2020, Proceedings of the 42nd International Conference on Software Engineering*, pages 1447–1458, Seoul, Korea, May 2020.

[28] Ellis Michael, Doug Woos, Thomas Anderson, Michael D. Ernst, and Zachary Tatlock. Teaching rigorous distributed systems with efficient model checking. In *EuroSys*, pages 1–15, Dresden, Germany, March 2019.

[29] Martin Kellogg, Vlastimil Dort, Suzanne Millstein, and Michael D. Ernst. Lightweight verification of array indexing. In *ISSTA 2018, Proceedings of the 2018 International Symposium on Software Testing and Analysis*, pages 3–14, Amsterdam, Netherlands, July 2018.

[30] Arianna Blasi, Alberto Goffi, Konstantin Kuznetsov, Alessandra Gorla, Michael D. Ernst, Mauro Pezzè, and Sergio Delgado Castellanos. Translating code comments to procedure specifications. In *ISSTA 2018, Proceedings of the 2018 International Symposium on Software Testing and Analysis*, pages 242–253, Amsterdam, Netherlands, July 2018.

[31] René Just, Chris Parnin, Ian Drosos, and Michael D. Ernst. Comparing developer-provided to user-provided tests for fault localization and automated program repair. In *ISSTA 2018, Proceedings of the 2018 International Symposium on Software Testing and Analysis*, pages 287–297, Amsterdam, Netherlands, July 2018.

[32] Pavel Panchekha, Adam Geller, Michael D. Ernst, Zachary Tatlock, and Shoaib Kamil. Verifying that web pages have accessible layout. In *PLDI 2018: Proceedings of the ACM SIGPLAN 2016 Conference on Programming Language Design and Implementation*, pages 1–14, Philadelphia, PA, USA, June 2018.

[33] Calvin Loncaric, Michael D. Ernst, and Emina Torlak. Generalized data structure synthesis. In *ICSE 2018, Proceedings of the 40th International Conference on Software Engineering*, pages 958–968, Gothenburg, Sweden, May 2018.

[34] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. NL2Bash: A corpus and semantic parser for natural language interface to the Linux operating system. In *LREC: Language Resources and Evaluation Conference*, Miyazaki, Japan, May 2018.

[35] Jonathan Jacky, Stefani Banerian, Michael D. Ernst, Calvin Loncaric, Stuart Pernsteiner, Zachary Tatlock, and Emina Torlak. Automatic formal verification for EPICS. In *ICALEPCS 2017: 16th International Conference on Accelerator and Large Experimental Physics Control Systems*, Barcelona, Spain, October 2017.

[36] Konstantin Weitz, Steven Lyubomirsky, Stefan Heule, Emina Torlak, Michael D. Ernst, and Zachary Tatlock. SpaceSearch: A library for building and verifying solver-aided tools. In *ICFP 2017: Proceedings of the 22nd ACM SIGPLAN International Conference on Functional Programming*, pages 25:1–25:28, Oxford, UK, September 2017.

[37] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D. Ernst, Deric Pang, and Benjamin Keller. Evaluating and improving fault localization. In *ICSE 2017, Proceedings of the 39th International Conference on Software Engineering*, pages 609–620, Buenos Aires, Argentina, May 2017.

[38] Michael D. Ernst. Natural language is a programming language: Applying natural language processing to software development. In *SNAPL 2017: the 2nd Summit oN Advances in Programming Languages*, pages 4:1–4:14, Asilomar, CA, USA, May 2017.

[39] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. Scalable verification of Border Gateway Protocol configurations with an SMT solver. In *OOPSLA 2016, Object-Oriented Programming Systems, Languages, and Applications*, pages 765–780, Amsterdam, November 2016.

[40] Chandrakana Nandi and Michael D. Ernst. Automatic trigger generation for rule-based smart homes. In *PLAS 2016: ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, pages 97–102, Vienna, Austria, October 2016.

[41] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. Formal semantics and automated verification for the Border Gateway Protocol. In *NetPL 2016: ACM SIGCOMM Workshop on Networking and Programming Languages (NetPL 2016)*, Florianópolis, Brazil, August 2016.

[42] Stuart Pernsteiner, Calvin Loncaric, Emina Torlak, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Jonathan Jacky. Investigating safety of a radiotherapy machine using system models with pluggable checkers. In *CAV 2016: 28th International Conference on Computer Aided Verification*, pages 23–41, Toronto, Canada, July 2016.

[43] Alberto Goffi, Alessandra Gorla, Michael D. Ernst, and Mauro Pezzè. Automatic generation of oracles for exceptional behaviors. In *ISSTA 2016, Proceedings of the 2016 International Symposium on Software Testing and Analysis*, pages 213–224, Saarbrücken, Genmany, July 2016.

[44] Calvin Loncaric, Emina Torlak, and Michael D. Ernst. Fast synthesis of fast collections. In *PLDI 2016: Proceedings of the ACM SIGPLAN 2016 Conference on Programming Language Design and Implementation*, pages 355–368, Santa Barbara, CA, USA, June 2016.

[45] Michael D. Ernst, Damiano Macedonio, Massimo Merro, and Fausto Spoto. Semantics for locking specifications. In *NFM 2016: 8th NASA Formal Methods Symposium*, pages 355–372, Minneapolis, MN, USA, June 2016.

[46] Michael D. Ernst, Alberto Lovato, Damiano Macedonio, Fausto Spoto, and Javier Thaine. Locking discipline inference and checking. In *ICSE 2016, Proceedings of the 38th International Conference on Software Engineering*, pages 1133–1144, Austin, TX, USA, May 2016.

[47] Doug Woos, James R. Wilcox, Steve Anton, Zachary Tatlock, Michael D. Ernst, and Thomas Anderson. Planning for change in a formal verification of the Raft consensus protocol. In *CPP 2016: 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 154–165, St. Petersburg, FL, USA, January 2016.

[48] Michael D. Ernst, Alberto Lovato, Damiano Macedonio, Ciprian Spiridon, and Fausto Spoto. Boolean formulas for the static identification of injection attacks in Java. In *LPAR 2015: Proceedings of the 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pages 130–145, Suva, Fiji, November 2015.

[49] Kıvanç Muşlu, Luke Swart, Yuriy Brun, and Michael D. Ernst. Development history granularity transformations. In *ASE 2015: Proceedings of the 30th Annual International Conference on Automated Software Engineering*, pages 697–702, Lincoln, NE, USA, November 2015.

[50] Paulo Barros, René Just, Suzanne Millstein, Paul Vines, Werner Dietl, Marcelo d'Amorim, and Michael D. Ernst. Static analysis of implicit control flow: Resolving Java reflection and Android intents. In *ASE 2015: Proceedings of the 30th Annual International Conference on Automated Software Engineering*, pages 669–679, Lincoln, NE, USA, November 2015.

[51] Brian Burg, Andrew J. Ko, and Michael D. Ernst. Explaining visual changes in web interfaces. In *UIST 2015: Proceedings of the 28th ACM Symposium on User Interface Software and Technology*, pages 259–268, Charlotte, NC, USA, November 2015.

[52] Irfan Ul Haq, Juan Caballero, and Michael D. Ernst. Ayudante: Identifying undesired variable interactions. In *WODA 2015: 13th International Workshop on Dynamic Analysis*, pages 8–13, Pittsburgh, PA, USA, October 2015.

[53] Drew Dean, Sean Guarino, Leonard Eusebi, Andrew Keplinger, Tim Pavlik, Ronald Watro, Aaron Cammarata, John Murray, Kelly McLaughlin, John Cheng, and Thomas Maddern. Lessons learned in game development for crowdsourced software formal verification. In *3GSE: USENIX Summit on Gaming, Games, and Gamification in Security Education*, Washington, DC, August 2015.

[54] Sai Zhang and Michael D. Ernst. Proactive detection of inadequate diagnostic messages for software configuration errors. In *ISSTA 2015, Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 12–23, Baltimore, MD, USA, July 2015.

[55] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas Anderson. Verdi: A framework for implementing and formally verifying distributed systems. In *PLDI 2015: Proceedings of the ACM SIGPLAN 2015 Conference on Programming Language Design and Implementation*, pages 357–368, Portland, OR, USA, June 2015.

[56] Mohsen Vakilian, Amarin Phaosawasdi, Michael D. Ernst, and Ralph E. Johnson. Cascade: A universal programmer-assisted type qualifier inference tool. In *ICSE 2015, Proceedings of the 37th International Conference on Software Engineering*, pages 234–245, Florence, Italy, May 2015.

[57] Michael D. Ernst, Dan Grossman, Jon Jacky, Calvin Loncaric, Stuart Pernsteiner, Zachary Tatlock, Emina Torlak, and Xi Wang. Toward a dependability case language and workflow for a radiation therapy system. In *SNAPL 2015: the Inaugural Summit oN Advances in Programming Languages*, pages 103–112, Asilomar, CA, USA, May 2015.

[58] Ruth Anderson, Michael D. Ernst, Robert Ordóñez, Paul Pham, and Ben Tribelhorn. A data programming CS1 course. In *SIGCSE: Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 150–155, Kansas City, MO, USA, March 2015.

[59] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *FSE 2014: Proceedings of the ACM SIGSOFT 22nd Symposium on the Foundations of Software Engineering*, pages 654–665, Hong Kong, November 2014.

[60] Michael D. Ernst, René Just, Suzanne Millstein, Werner Dietl, Stuart Pernsteiner, Franziska Roesner, Karl Koscher, Paulo Barros, Ravi Bhoraskar, Seungyeop Han, Paul Vines, and Edward X. Wu. Collaborative verification of information flow for a high-assurance app store. In *CCS 2014: Proceedings of the 21st ACM Conference on Computer and Communications Security*, pages 1092–1104, Scottsdale, AZ, USA, November 2014.

[61] René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A Database of existing faults to enable controlled testing studies for Java programs. In *ISSTA 2014, Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 437–440, San Jose, CA, USA, July 2014. Tool demo.

[62] René Just, Michael D. Ernst, and Gordon Fraser. Efficient mutation analysis by propagating and partitioning infected execution states. In *ISSTA 2014, Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 315–326, San Jose, CA, USA, July 2014.

[63] Konstantin Weitz, Gene Kim, Siwakorn Srisakaokul, and Michael D. Ernst. A type system for format strings. In *ISSTA 2014, Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 127–137, San Jose, CA, USA, July 2014.

[64] Sai Zhang, Darioush Jalali, Jochen Wuttke, Kıvanç Muşlu, Wing Lam, Michael D. Ernst, and David Notkin. Empirically revisiting the test independence assumption. In *ISSTA 2014, Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 385–396, San Jose, CA, USA, July 2014.

[65] Ravi Bhoraskar, Dominic Langenegger, Pingyang He, Raymond Cheng, Will Scott, and Michael D. Ernst. User scripting on Android using BladeDroid. In *APSys 2014: 5th Asia-Pacific Workshop on Systems*, pages 9:1–9:7, Beijing, China, June 2014.

[66] Jenny Abrahamson, Ivan Beschastnikh, Yuriy Brun, and Michael D. Ernst. Shedding light on distributed system executions. In *ICSE 2014, Proceedings of the 36th International Conference on Software Engineering*, pages 598–599, Hyderabad, India, June 2014.

[67] Todd W. Schiller, Kellen Donohue, Forrest Coward, and Michael D. Ernst. Case studies and tools for contract specifications. In *ICSE 2014, Proceedings of the 36th International Conference on Software Engineering*, pages 596–607, Hyderabad, India, June 2014.

[68] Sai Zhang and Michael D. Ernst. Which configuration option should I change? In *ICSE 2014, Proceedings of the 36th International Conference on Software Engineering*, pages 152–163, Hyderabad, India, June 2014.

[69] Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst, and Arvind Krishnamurthy. Inferring models of concurrent systems from logs of their behavior with CSight. In *ICSE 2014, Proceedings of the 36th International Conference on Software Engineering*, pages 468–479, Hyderabad, India, June 2014.

[70] Ruth Anderson, Michael D. Ernst, Robert Ordóñez, Paul Pham, and Steven A. Wolfman. Introductory programming meets the real world: Using real problems and data in CS1. In *SIGCSE: Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 465–466, Atlanta, GA, USA, March 2014.

[71] Brian Burg, Richard Bailey, Andrew J. Ko, and Michael D. Ernst. Interactive record/replay for web application debugging. In *UIST 2013: Proceedings of the 26th ACM Symposium on User Interface Software and Technology*, pages 473–484, St. Andrews, UK, October 2013.

[72] Kıvanç Muşlu, Yuriy Brun, Michael D. Ernst, and David Notkin. Making offline analyses continuous. In *ESEC/FSE 2013: The 9th joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 323–333, St. Petersburg, Russia, August 2013.

[73] Sai Zhang, Hao Lü, and Michael D. Ernst. Automatically repairing broken workflows for evolving GUI applications. In *ISSTA 2013, Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pages 45–55, Lugano, Switzerland, July 2013.

[74] Colin S. Gordon, Werner Dietl, Michael D. Ernst, and Dan Grossman. JavaUI: Effects for controlling UI object access. In *ECOOP 2013 — Object-Oriented Programming, 27th European Conference*, pages 179–204, Montpellier, France, July 2013.

[75] Colin S. Gordon, Michael D. Ernst, and Dan Grossman. Rely-guarantee references for refinement types over aliased mutable data. In *PLDI 2013: Proceedings of the ACM SIGPLAN 2013 Conference on Programming Language Design and Implementation*, pages 73–84, Seattle, WA, USA, June 2013.

[76] Alex Potanin, Johan Östlund, Yoav Zibin, and Michael D. Ernst. Immutability. In *Aliasing in Object-Oriented Programming*, volume 7850 of *LNCS*, pages 233–269. Springer-Verlag, April 2013.

[77] Ivan Beschastnikh, Yuriy Brun, Jenny Abrahamson, Michael D. Ernst, and Arvind Krishnamurthy. Unifying FSM-inference algorithms through declarative specification. In *ICSE 2013, Proceedings of the 35th International Conference on Software Engineering*, pages 252–261, San Francisco, CA, USA, May 2013.

[78] Sai Zhang and Michael D. Ernst. Automated diagnosis of software configuration errors. In *ICSE 2013, Proceedings of the 35th International Conference on Software Engineering*, pages 312–321, San Francisco, CA, USA, May 2013.

[79] Wei Huang, Ana Milanova, Werner Dietl, and Michael D. Ernst. ReIm & ReImInfer: Checking and inference of reference immutability and method purity. In *OOPSLA 2012, Object-Oriented Programming Systems, Languages, and Applications*, pages 879–896, Tucson, AZ, USA, October 2012.

[80] Todd W. Schiller and Michael D. Ernst. Reducing the barriers to writing verified specifications. In *OOPSLA 2012, Object-Oriented Programming Systems, Languages, and Applications*, pages 95–112, Tucson, AZ, USA, October 2012.

[81] Kıvanç Muşlu, Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Speculative analysis of integrated development environment recommendations. In *OOPSLA 2012, Object-Oriented Programming Systems, Languages, and Applications*, pages 669–682, Tucson, AZ, USA, October 2012.

[82] Sai Zhang, Hao Lü, and Michael D. Ernst. Finding errors in multithreaded GUI applications. In *ISSTA 2012, Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 243–253, Minneapolis, MN, USA, July 2012.

[83] Wei Huang, Werner Dietl, Ana Milanova, and Michael D. Ernst. Inference and checking of object ownership. In *ECOOP 2012 — Object-Oriented Programming, 26th European Conference*, pages 181–206, Beijing, China, June 2012.

[84] Werner Dietl, Stephanie Dietzel, Michael D. Ernst, Nathaniel Mote, Brian Walker, Seth Cooper, Timothy Pavlik, and Zoran Popović. Verification games: Making verification fun. In *FTfJP: 14th Workshop on Formal Techniques for Java-like Programs*, pages 42–49, Beijing, China, June 2012.

[85] Eric Spishak, Werner Dietl, and Michael D. Ernst. A type system for regular expressions. In *FTfJP: 14th Workshop on Formal Techniques for Java-like Programs*, pages 20–26, Beijing, China, June 2012.

[86] Jingyue Li and Michael D. Ernst. CBCD: Cloned Buggy Code Detector. In *ICSE 2012, Proceedings of the 34th International Conference on Software Engineering*, pages 310–320, Zürich, Switzerland, June 2012.

[87] Yuriy Brun, Kıvanç Muşlu, Reid Holmes, Michael D. Ernst, and David Notkin. Predicting development trajectories to prevent collaboration conflicts. In *FutureCSD 2012: The Future of Collaborative Software Development*, Bellevue, WA, USA, February 2012.

[88] Colin S. Gordon, Michael D. Ernst, and Dan Grossman. Static lock capabilities for deadlock freedom. In *TLDI 2012: The seventh ACM SIGPLAN Workshop on Types in Language Design and Implementation*, pages 67–78, Philadelphia, PA, USA, January 2012.

[89] Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst, Arvind Krishnamurthy, and Thomas E. Anderson. Bandsaw: Log-powered test scenario generation for distributed systems. In *SOSP WIP: Proceedings of the 23rd ACM Symposium on Operating Systems Principles, Work In Progress Track*, Cascais, Portugal, October 2011.

[90] Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst, Arvind Krishnamurthy, and Thomas E. Anderson. Mining temporal invariants from partially ordered logs. In *SLAML 2011: Workshop on Managing Large-Scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques (SLAML '11)*, Cascais, Portugal, October 2011. Article No. 3.

[91] Brian Robinson, Michael D. Ernst, Jeff H. Perkins, Vinay Augustine, and Nuo Li. Scaling up automated test generation: Automatically generating maintainable regression unit tests for programs. In *ASE 2011: Proceedings of the 26th Annual International Conference on Automated Software Engineering*, pages 23–32, Lawrence, KS, USA, November 2011.

[92] Sai Zhang, Cheng Zhang, and Michael D. Ernst. Automated documentation inference to explain failed tests. In *ASE 2011: Proceedings of the 26th Annual International Conference on Automated Software Engineering*, pages 63–72, Lawrence, KS, USA, November 2011.

[93] Werner Dietl, Michael D. Ernst, and Peter Müller. Tunable static inference for Generic Universe Types. In *ECOOP 2011 — Object-Oriented Programming, 25th European Conference*, pages 333–357, Lancaster, UK, July 2011.

[94] Sai Zhang, David Saff, Yingyi Bu, and Michael D. Ernst. Combined static and dynamic automated test generation. In *ISSTA 2011, Proceedings of the 2011 International Symposium on Software Testing and Analysis*, pages 353–363, Toronto, Canada, July 2011.

[95] Werner Dietl, Stephanie Dietzel, Michael D. Ernst, Kıvanç Muşlu, and Todd Schiller. Building and using pluggable type-checkers. In *ICSE 2011, Proceedings of the 33rd International Conference on Software Engineering*, pages 681–690, Waikiki, Hawaii, USA, May 2011.

[96] Michael Bayne, Richard Cook, and Michael D. Ernst. Always-available static and dynamic feedback. In *ICSE 2011, Proceedings of the 33rd International Conference on Software Engineering*, pages 521–530, Waikiki, Hawaii, USA, May 2011.

[97] Fausto Spoto and Michael D. Ernst. Inference of field initialization. In *ICSE 2011, Proceedings of the 33rd International Conference on Software Engineering*, pages 231–240, Waikiki, Hawaii, USA, May 2011.

[98] Danny Dig, John Marrero, and Michael D. Ernst. How do programs become more concurrent? a story of program transformations. In *Proceedings of the 4th International Workshop on Multicore Software Engineering*, pages 43–50, Waikiki, Hawaii, USA, May 2011.

[99] Todd W. Schiller and Michael D. Ernst. Rethinking the economics of software engineering. In *FoSER: Workshop on the Future of Software Engineering Research*, pages 325–330, Santa Fe, NM, USA, November 2010.

[100] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Speculative analysis: Exploring future development states of software. In *FoSER: Workshop on the Future of Software Engineering Research*, pages 59–64, Santa Fe, NM, USA, November 2010.

[101] Yoav Zibin, Alex Potanin, Paley Li, Mahmood Ali, and Michael D. Ernst. Ownership and immutability in generic Java. In *OOPSLA 2010, Object-Oriented Programming Systems, Languages, and Applications*, pages 598–617, Revo, NV, USA, October 2010.

[102] Sigurd Schneider, Ivan Beschastnikh, Slava Chernyak, Michael D. Ernst, and Yuriy Brun. Synoptic: Summarizing system logs with refinement. In *SLAML 2010: Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML '10)*, Vancouver, BC, Canada, October 2010.

[103] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. HaLoop: Efficient iterative data processing on large clusters. In *VLDB 2010: 36th International Conference on Very Large Data Bases*, pages 285–296, Singapore, September 2010.

[104] Adam Kieżun, Philip J. Guo, Karthick Jayaraman, and Michael D. Ernst. Automatic creation of SQL injection and cross-site scripting attacks. In *ICSE 2009, Proceedings of the 31st International Conference on Software Engineering*, pages 199–209, Vancouver, BC, Canada, May 2009.

[105] Danny Dig, John Marrero, and Michael D. Ernst. Refactoring sequential Java code for concurrency via concurrent libraries. In *ICSE 2009, Proceedings of the 31st International Conference on Software Engineering*, pages 397–407, Vancouver, BC, Canada, May 2009.

[106] Matthew M. Papi, Mahmood Ali, Telmo Luis Correa Jr., Jeff H. Perkins, and Michael D. Ernst. Practical pluggable types for Java. In *ISSTA 2008, Proceedings of the 2008 International Symposium on Software Testing and Analysis*, pages 201–212, Seattle, WA, USA, July 2008.

[107] Shay Artzi, Sunghun Kim, and Michael D. Ernst. Recrash: Making software failures reproducible by preserving object states. In *ECOOP 2008 — Object-Oriented Programming, 22nd European Conference*, pages 542–565, Paphos, Cyprus, July 2008.

[108] Jaime Quinonez, Matthew S. Tschantz, and Michael D. Ernst. Inference of reference immutability. In *ECOOP 2008 — Object-Oriented Programming, 22nd European Conference*, pages 616–641, Paphos, Cyprus, July 2008.

[109] Stephen McCamant and Michael D. Ernst. Quantitative information flow as network flow capacity. In *PLDI 2008: Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation*, pages 193–205, Tucson, AZ, USA, June 2008.

[110] Sunghun Kim and Michael D. Ernst. Which warnings should I fix first? In *ESEC/FSE 2007: Proceedings of the 11th European Software Engineering Conference and the 15th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 45–54, Dubrovnik, Croatia, September 2007.

[111] Yoav Zibin, Alex Potanin, Mahmood Ali, Shay Artzi, Adam Kieżun, and Michael D. Ernst. Object and reference immutability using Java generics. In *ESEC/FSE 2007: Proceedings of the 11th European Software Engineering Conference and the 15th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 75–84, Dubrovnik, Croatia, September 2007.

[112] Stephen McCamant and Michael D. Ernst. A simulation-based proof technique for dynamic information flow. In *PLAS 2007: ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, pages 41–46, San Diego, California, USA, June 2007.

[113] Sunghun Kim and Michael D. Ernst. Prioritizing warnings by analyzing software history. In *MSR 2007: International Workshop on Mining Software Repositories*, pages 27–30, Minneapolis, MN, USA, May 2007.

[114] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. Feedback-directed random test generation. In *ICSE 2007, Proceedings of the 29th International Conference on Software Engineering*, pages 75–84, Minneapolis, MN, USA, May 2007.

[115] Adam Kieżun, Michael D. Ernst, Frank Tip, and Robert M. Fuhrer. Refactoring for parameterizing Java classes. In *ICSE 2007, Proceedings of the 29th International Conference on Software Engineering*, pages 437–446, Minneapolis, MN, USA, May 2007.

[116] Michael D. Ernst. The Groupthink specification exercise. In *ICSE Education and Training Track: Software Engineering Education in the Modern Age: Challenges and Possibilities, PostProceedings of ICSE '05 Education and Training Track*, volume 4309 of *Lecture Notes in Computer Science*, pages 89–107. Springer, St. Louis, MO, USA, December 2006.

[117] Shay Artzi, Michael D. Ernst, Adam Kieżun, Carlos Pacheco, and Jeff H. Perkins. Finding the needles in the haystack: Generating legal test inputs for object-oriented programs. In *M-TOOS: 1st Workshop*

on Model-Based Testing and Object-Oriented Systems, pages 27–34, Portland, OR, USA, October 2006.

[118] Marcelo d'Amorim, Carlos Pacheco, Darko Marinov, Tao Xie, and Michael D. Ernst. An empirical comparison of automated generation and classification techniques for object-oriented unit testing. In *ASE 2006: Proceedings of the 21st Annual International Conference on Automated Software Engineering*, pages 59–68, Tokyo, Japan, September 2006.

[119] Philip J. Guo, Jeff H. Perkins, Stephen McCamant, and Michael D. Ernst. Dynamic inference of abstract types. In *ISSTA 2006, Proceedings of the 2006 International Symposium on Software Testing and Analysis*, pages 255–265, Portland, ME, USA, July 2006.

[120] Brian Demsky, Michael D. Ernst, Philip J. Guo, Stephen McCamant, Jeff H. Perkins, and Martin Rinard. Inference and enforcement of data structure consistency specifications. In *ISSTA 2006, Proceedings of the 2006 International Symposium on Software Testing and Analysis*, pages 233–243, Portland, ME, USA, July 2006.

[121] Michael D. Ernst, Raimondas Lencevicius, and Jeff H. Perkins. Detection of web service substitutability and composability. In *WS-MaTe: International Workshop on Web Services — Modeling and Testing*, pages 123–135, Palermo, Italy, June 2006.

[122] David Saff, Shay Artzi, Jeff H. Perkins, and Michael D. Ernst. Automatic test factoring for Java. In *ASE 2005: Proceedings of the 20th Annual International Conference on Automated Software Engineering*, pages 114–123, Long Beach, CA, USA, November 2005.

[123] Shay Artzi and Michael D. Ernst. Using predicate fields in a highly flexible industrial control system. In *OOPSLA 2005, Object-Oriented Programming Systems, Languages, and Applications*, pages 319–330, San Diego, CA, USA, October 2005.

[124] Matthew S. Tschantz and Michael D. Ernst. Javari: Adding reference immutability to Java. In *OOPSLA 2005, Object-Oriented Programming Systems, Languages, and Applications*, pages 211–230, San Diego, CA, USA, October 2005.

[125] Michael D. Ernst. Verification for legacy programs. In *VSTTE 2005: Verified Software: Theories, Tools, Experiments*, Zürich, Switzerland, October 2005.

[126] Amy Williams, William Thies, and Michael D. Ernst. Static deadlock detection for Java libraries. In *ECOOP 2005 — Object-Oriented Programming, 19th European Conference*, pages 602–629, Glasgow, Scotland, July 2005.

[127] Carlos Pacheco and Michael D. Ernst. Eclat: Automatic generation and classification of test inputs. In *ECOOP 2005 — Object-Oriented Programming, 19th European Conference*, pages 504–527, Glasgow, Scotland, July 2005.

[128] Jeff H. Perkins and Michael D. Ernst. Efficient incremental algorithms for dynamic detection of likely invariants. In *FSE 2004: Proceedings of the ACM SIGSOFT 12th Symposium on the Foundations of Software Engineering*, pages 23–32, Newport Beach, CA, USA, November 2004.

[129] Stephen McCamant and Michael D. Ernst. Formalizing lightweight verification of software component composition. In *SAVCBS 2004: Specification and Verification of Component-Based Systems*, pages 47–54, Newport Beach, CA, USA, October 2004.

[130] Alan Donovan, Adam Kieżun, Matthew S. Tschantz, and Michael D. Ernst. Converting Java programs to use generic libraries. In *OOPSLA 2004, Object-Oriented Programming Systems, Languages, and Applications*, pages 15–34, Vancouver, BC, Canada, October 2004.

[131] Adrian Birka and Michael D. Ernst. A practical type system and language for reference immutability. In *OOPSLA 2004, Object-Oriented Programming Systems, Languages, and Applications*, pages 35–49, Vancouver, BC, Canada, October 2004.

[132] Lee Lin and Michael D. Ernst. Improving adaptability via program steering. In *ISSTA 2004, Proceedings of the 2004 International Symposium on Software Testing and Analysis*, pages 206–216, Boston, MA, USA, July 2004.

[133] David Saff and Michael D. Ernst. An experimental evaluation of continuous testing during development. In *ISSTA 2004, Proceedings of the 2004 International Symposium on Software Testing and Analysis*, pages 76–85, Boston, MA, USA, July 2004.

[134] Stephen McCamant and Michael D. Ernst. Early identification of incompatibilities in multi-component upgrades. In *ECOOP 2004 — Object-Oriented Programming, 18th European Conference*, pages 440–464, Oslo, Norway, June 2004.

[135] David Saff and Michael D. Ernst. Automatic mock object creation for test factoring. In *PASTE 2004: ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'04)*, pages 49–51, Washington, DC, USA, June 2004.

[136] Yuriy Brun and Michael D. Ernst. Finding latent code errors via machine learning over program executions. In *ICSE 2004, Proceedings of the 26th International Conference on Software Engineering*, pages 480–490, Edinburgh, Scotland, May 2004.

[137] David Saff and Michael D. Ernst. Continuous testing in Eclipse. In *2nd Eclipse Technology Exchange Workshop (eTX)*, Barcelona, Spain, March 2004.

[138] David Saff and Michael D. Ernst. Reducing wasted development time via continuous testing. In *ISSRE 2003: Fourteenth International Symposium on Software Reliability Engineering*, pages 281–292, Denver, CO, November 2003.

[139] Stephen McCamant and Michael D. Ernst. Predicting problems caused by component upgrades. In *ESEC/FSE 2003: Proceedings of the 9th European Software Engineering Conference and the 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 287–296, Helsinki, Finland, September 2003.

[140] Michael D. Ernst. Static and dynamic analysis: Synergy and duality. In *WODA 2003: Workshop on Dynamic Analysis*, pages 24–27, Portland, OR, USA, May 2003.

[141] Michael Harder, Jeff Mellen, and Michael D. Ernst. Improving test suites via operational abstraction. In *ICSE 2003, Proceedings of the 25th International Conference on Software Engineering*, pages 60–71, Portland, Oregon, May 2003.

[142] Jeremy W. Nimmer and Michael D. Ernst. Invariant inference for static checking: An empirical evaluation. In *FSE 2002, Proceedings of the ACM SIGSOFT 10th International Symposium on the Foundations of Software Engineering*, pages 11–20, Charleston, SC, November 2002.

[143] Jeremy W. Nimmer and Michael D. Ernst. Automatic generation of program specifications. In *ISSTA 2002, Proceedings of the 2002 International Symposium on Software Testing and Analysis*, pages 232–242, Rome, Italy, July 2002.

[144] Yoshio Kataoka, Michael D. Ernst, William G. Griswold, and David Notkin. Automated support for program refactoring using invariants. In *ICSM 2001: Proceedings of the International Conference on Software Maintenance*, pages 736–743, Florence, Italy, November 2001.

[145] Jeremy W. Nimmer and Michael D. Ernst. Static verification of dynamically detected program invariants: Integrating Daikon and ESC/Java. In *RV 2001: Proceedings of the First Workshop on Runtime Verification*, Paris, France, July 2001.

[146] Michael D. Ernst, Adam Czeisler, William G. Griswold, and David Notkin. Quickly detecting relevant program invariants. In *ICSE 2000, Proceedings of the 22nd International Conference on Software Engineering*, pages 449–458, Limerick, Ireland, June 2000.

[147] Michael D. Ernst, Craig S. Kaplan, and Craig Chambers. Predicate dispatching: A unified theory of dispatch. In *ECOOP '98: the 12th European Conference on Object-Oriented Programming*, pages 186–211, Brussels, Belgium, July 1998.

[148] Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-compilation of planning problems. In *IJCAI '97: IJCAI-97, Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1169–1176, Nagoya, Aichi, Japan, August 1997.

[149] Daniel Weise, Roger F. Crew, Michael D. Ernst, and Bjarne Steensgaard. Value dependence graphs: Representation without taxation. In *POPL '94: Proceedings of the 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 297–310, Portland, OR, January 1994.

[150] Michael D. Ernst and Bruce E. Flinchbaugh. Image/map correspondence using curve matching. In *AAAI Spring Symposium on Robot Navigation*, Stanford, CA, March 28–30, 1989. Also published as Texas Instruments Technical Report CSC-SIUL-89-12.

[151] Michael D. Ernst. Ml typechecking is not efficient. In *Papers of the MIT ACM Undergraduate Conference*, April 1989.

Conference proceedings and other non-journal articles – Refereed by abstract only Not applicable.

Complete books written Not applicable.

Parts of books (chapters in edited books) Not applicable.

Books edited Not applicable.

Proceedings and journal issues edited

[152] Michael D. Ernst and Thomas Jensen, editors. *PASTE: ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, Lisbon, Portugal, September 2005.

[153] Jonathan E. Cook and Michael D. Ernst, editors. *WODA 2003: ICSE Workshop on Dynamic Analysis*, Portland, Oregon, May 2003.

[154] Michael D. Ernst, editor. *IR '95: Intermediate Representations Workshop Proceedings*, San Francisco, CA, January 22, 1995. *ACM SIGPLAN Notices* 30(3), March 1995.

[155] Michael D. Ernst. Intellectual property in computing: (How) should software be protected? An industry perspective. Memo AIM-1369, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, May 1992.

Patents submitted and/or awarded

[156] Gideon A. Yuval and Michael D. Ernst. Method and system for controlling unauthorized access to information distributed to users. U.S. Patent 5,586,186, December 17, 1996. Assigned to Microsoft Corporation.

Abstracts, letters, non-refereed papers, technical reports

[157] Michael D. Ernst. Type Annotations specification (JSR 308). `https://checkerframework.org/jsr308/`, October 2011.

[158] David Saff, Marat Boshernitsan, and Michael D. Ernst. Theories in practice: Easy-to-write specifications that catch bugs. Technical Report MIT-CSAIL-TR-2008-002, MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, January 14, 2008.

[159] Michael D. Ernst and Jeff H. Perkins. Learning from executions: Dynamic analysis for software engineering and program understanding, November 2005. Tutorial at ASE 2005.

[160] Nii Dodoo, Lee Lin, and Michael D. Ernst. Selecting, refining, and evaluating predicates for program analysis. Technical Report MIT-LCS-TR-914, MIT Laboratory for Computer Science, Cambridge, MA, July 21, 2003.

[161] Samir V. Meghani and Michael D. Ernst. Determining legal method call sequences in object interfaces. `https://homes.cs.washington.edu/~mernst/pubs/call-sequences.pdf`, May 2003.

[162] David Notkin, Marc Donner, Michael D. Ernst, Michael Gorlick, and E. James Whitehead, Jr. Panel: Perspectives on software engineering. In *ICSE 2001, Proceedings of the 23rd International Conference on Software Engineering*, pages 699–702, Montreal, Canada, May 2001.

[163] Michael D. Ernst. *Dynamically Discovering Likely Program Invariants*. PhD thesis, University of Washington Department of Computer Science and Engineering, Seattle, Washington, August 2000.

[164] Michael D. Ernst, William G. Griswold, Yoshio Kataoka, and David Notkin. Dynamically discovering pointer-based program invariants. Technical Report UW-CSE-99-11-02, University of Washington Department of Computer Science and Engineering, Seattle, WA, November 16, 1999. Revised March 17, 2000.

[165] Michael D. Ernst. Slicing pointers and procedures (abstract). Technical Report MSR-TR-95-23, Microsoft Research, Redmond, WA, January 13, 1995.

[166] Michael D. Ernst. Serializing parallel programs by removing redundant computation. Technical Report MIT/LCS/TR-638, MIT Laboratory for Computer Science, Cambridge, MA, August 21, 1994.

[167] Michael D. Ernst. Practical fine-grained static slicing of optimized code. Technical Report MSR-TR-94-14, Microsoft Research, Redmond, WA, July 26, 1994.

[168] Michael D. Ernst and Gideon Yuval. Heraclitean encryption. Technical Report MSR-TR-94-13, Microsoft Research, Redmond, WA, March 3, 1994.

[169] Michael D. Ernst. Adequate models for recursive program schemes. Bachelors thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1989.

Other significant research dissemination (web sites, software, Wikis, etc.)

In order to permit others to reproduce my results and build upon my research, I make all my research artifacts publicly available (most linked from `https://homes.cs.washington.edu/~mernst/software/`, along with more information about each; and others by request). Some of the more significant research systems that are used by others include the following. Most of these were created in collaboration with my students and other colleagues.

1. Daikon dynamically detects likely program invariants in C, C++, Java, Perl, and other languages. Almost 100 papers (that I know of; see `http://plse.cs.washington.edu/daikon/pubs/#daikon-methodology`) use Daikon as an integral part of their research methodology, and additional papers use Daikon as a test subject Daikon is being used internally by a number of companies, and is being commercialized by Agitar, Determina, and Scrutiny (that I know of). Regarding Agitar's outgrowth from Daikon, see their ISSTA 2006 paper "From Daikon to Agitator: Lessons and Challenges in Building a Commercial Tool for Developer Testing". Agitar's awards include Java One Duke's Choice, Jolt award, the Wall Street Journal Software Technology Innovation Award, and InfoWorld Technology of the Year.
2. The generics refactoring in Eclipse — the most widely-used Java integrated development environment (IDE) — is an adaptation of our sound, precise, and scalable systems that introduce parametric polymorphism (generic types) in Java programs.
3. Continuous testing augments an IDE to rapidly indicate semantic errors, just as they already do for syntactic errors. The key idea is to run tests in the background and indicate test failures. Careful user interface design results in a system that helps — not annoys — programmers, as indicated by case studies and controlled experiments. Continuous testing is a popular plug-in for Eclipse.
4. Version 4 of the popular and widely-used JUnit regression testing framework was built in part in my group, by David Saff (in conjunction with Kent Beck and Erich Gamma). The JUnit plug-in distributed with Eclipse (version 3.2 and later) was built in my group, by David Saff and others. It replaced a previous version built by the Eclipse team that had lesser functionality.

5. Test factoring is a capture–replay technique for converting a long-running system test into a collection of fast unit tests that exercise software components in the same way the system test did. Our implementation scales to programs of hundreds of thousands of lines, and others have since produced their own implementations.
6. The Eclat, Joe, Randoop, and Palulu systems automatically generate test inputs, using novel techniques to create relevant and effective tests. Collectively, these four systems have found hundreds of errors in the Sun and IBM Java Development Kits (JDKs), and in the Microsoft equivalent (the Common Language Runtime), which are heavily-tested, widely-deployed commercial infrastructure.
7. The Checker Framework allows easy creation of pluggable type-checkers. A pluggable type-checker is a compiler plug-in that extends and strengthens the type system of a programming language. As a result, the compiler can detect and prevent errors that would otherwise have caused incorrect behavior at run time. We built over a dozen type-checkers using this framework, and research groups around the world have adopted the Checker Framework to build many more. This work was inspired by the difficulty we experienced when creating specialized type-checkers, such as that we built for Javari.
8. Synoptic mines system logs and produces a model of observed system behavior, to help programmers to understand their systems and fix bugs, improve performance, or perform maintenance.
9. Eclipse's Quick Fix facility suggests ways to fix a compilation errors. Kıvanç Muşlu's Quick Fix Scout plug-in augments the Quick Fix menu by indicating the effect of each suggestion (how many compilation errors it fixes), adding better suggestions, and sorting all the suggestions. It does this by actually trying each suggestion in the background.
10. The Crystal tool proactively warns when two developers make conflicting changes to a codebase, before the developers have shared their changes with one another. By utilizing local commits and speculative analysis, it suffers no false positives, unlike other tools. Microsoft has re-implemented this tool as Beacon, and is deploying Beacon internally.
11. A static ownership inference system (for Generic Universe Types) enables tuning by users to find the best among many possible legal ownership structures, using a Max-SAT solver.
12. The Ductile system combines the benefits of static and dynamic typing, allowing a programmer to utilize sound static typing or fully dynamic execution, using the same codebase and switching between the two on demand with no program edits required.
13. HaLoop makes the popular Hadoop Map–Reduce implementation more efficient, by caching and re-using results in iterative computations.
14. The HAMPI solver for string constraints is used in a variety of security and verification contexts.
15. The ClearView system automatically detects security attacks and bugs, proposes and evaluates fixes that inoculate the system from the attacks/bugs, and deploys the best fix — all without human intervention and in seconds.
16. We have extended Java's annotation system, and Oracle plans to include our extension (in Java parlance, "JSR 308") in the Java 8 language and use our implementation in the Java Development Kit (JDK).
17. The Fjalar toolkit enables instrumentation of compiled executables (currently for Linux/x86). It combines the benefits of binary and source rewriting through a "mixed-level" approach. It is the basis for a value profiling tool (Kvasir) and an abstract type inference (DynComp).
18. DynComp infers abstract types for Java, C, and C++ programs, retrieving structure from source code in which a single type is used to represent multiple concepts.
19. The Groupthink Specification Exercise is a fun group activity that teaches students the value of specifications, the difficulty of creating them, and various approaches for doing so. It has been used worldwide. I distribute, handouts, lecture slides, instructors' notes, and optional software that automates

running the activity.

20. Stephen McCamant's PittSFIeld tool performs efficient sandboxing for the x86 architecture, which presents challenges (such as variable-length instructions) over RISC systems for which sandboxing had been previously applied.
21. Medic was the first tool for compiling a problem into SAT, or logical satisfiability. (The idea was due to Kautz and Selman, but this is the first implementation.) Such a transformation is valuable because of engineering and research that has made SAT-solving very fast. Today, solving problems by reducing them to SAT is a very common technique used in all fields of computer science.
22. The Gamesman's Toolkit facilitates combinatorial game-theoretic analysis of complete-information games. I extended it with support for analyzing the ancient Hawaiian game of Konane.
23. Subject programs from my experiments are often requested by other researchers who do not wish to go to the trouble of collecting and organizing realistic programs that can be used to validate research in testing and other areas of software engineering.

Other software systems stemming from my research that are used by others include cppp (a partial evaluator for cpp, the C preprocessor), Gud (a prototype implementation of predicate dispatching), and contributions to the IOA toolkit distributed by Nancy Lynch's Theory of Distributed Systems group. Educational contributions include infrastructure for supporting programming classes. Widely used software not stemming from research includes EDB (a database system), contributions to Emacs and other free software, bibtex2web (software for creating web pages from BibTEX bibliography files), and others. I also maintain a popular collection of advice for students and researchers, at `http://homes.cs.washington.edu/~mernst/advice/`.

OTHER SCHOLARLY ACTIVITY

Invited lectures and seminars

Only talks since 2000 are listed. Talks at closed events (e.g., PI meetings, UW events) are not listed. Talks given in connection with a refereed publication are not listed. Talks given by co-authors are not listed.

1. SCAM, "Custom type-checking for programming, teaching, and research", September 28, 2020
2. Amazon Web Services, "Implement your own type system in 1 hour", March 20, 2019
3. ISSTA, "Pluggable type systems reconsidered", Impact Paper Award talk, July 2018
4. Code One "Implement Your Own Type System in 45 Minutes", October 2018
5. Code One "Preventing Errors Before They Happen: The Checker Framework", October 2018
6. Code One "Using Type Annotations to Improve Your Code", October 2018
7. ISSTA, "Comparing developer-provided to user-provided tests for fault localization and automated program repair", July 2018
8. Oracle, "Create your own type system in 1 hour", May 16, 2018
9. Seattle Java User's Group, "Create your own type system in 1 hour", February 20, 2018
10. JavaOne, "Preventing Errors Before They Happen", Oct 2, 2017
11. JavaOne, "Using Type Annotations to Improve Your Code", Oct 2, 2017
12. Open Source Bridge, "Create your own type system in 45 minutes", June 22, 2017
13. ETAPS keynote, "Natural language is a programming language", April 27, 2017

14. TU Delft, "Preventing null pointer exceptions at compile time", April 25, 2017
15. TU Delft, "Natural language is a programming language", April 25, 2017
16. CWI Amsterdam (Centrum Wiskunde & Informatica) "Preventing errors before they happen", April 24, 2017
17. Devoxx, "Preventing Null pointer exceptions at compile time", March 22, 2017
18. Devoxx, "The Checker Framework in action: Preventing errors kefore they happen", March 21, 2017
19. Georgia Institute of Technology, "Natural language is a programming language", January 24, 2017
20. Triangle Computer Science Distinguished Lecturer Series, "Natural language is a programming language", January 23, 2017
21. Microsoft Research, "Analyzing the entire program: applying natural language processing to software engineering", January 11, 2017
22. University of Southern California, "Analyzing the entire program: applying natural language processing to software engineering", December 1, 2016
23. JavaOne, "Preventing Errors Before They Happen", September 19, 2016
24. JavaOne, "Disciplined Locking: No More Concurrency Errors", September 19, 2016
25. JavaOne, "Using Type Annotations to Improve Your Code", September 19, 2016
26. Pacific Northwest Programming Languages and Software Engineering Meeting, "Natural language processing meets software testing", March 15, 2016
27. University of Washington at Bothell, "Preventing errors before they happen with pluggable type-checking", November 16, 2015
28. JavaOne, "Preventing Errors Before They Happen", October 26, 2015
29. JavaOne, "Collaborative Verification of the Information Flow for a High-Assurance App Store", October 26, 2015
30. JavaOne, "Using Type Annotations to Improve Your Code", October 26, 2015
31. University of Buenos Aires, "Preventing errors before they happen: Lightweight verification via pluggable type-checking", June 5, 2015
32. University of Buenos Aires, "Collaborative verification of information flow for a high-assurance app store", June 1, 2015
33. CSI-SE, (keynote), May 19, 2015
34. CIbSE, "Lightweight software verification with pluggable type-checking", April 23, 2015
35. University of Buenos Aires, "Verification games: Making software verification fun", April 13, 2015
36. IMDEA Software Institute, "Collaborative verification of information flow for a high-assurance app store", November 18, 2014
37. Jazoon, "GUI threading explained and verified", October 21, 2014
38. Coverity, "The Checker Framework: Pluggable static analysis for Java", August 27, 2014
39. SIGCSE, "Introductory programming meets the real world: Using real problems and data in CS1", March 7, 2014
40. Georgia Institute of Technology, "Verification games: Making software verification fun", March 6, 2014
41. University of California at San Diego, "Collaborative verification of information flow for a high-assurance app store", January 28, 2014
42. University of Texas, "Verification games: Making software verification fun", December 2, 2013
43. University of Texas, "Collaborative verification of information flow for a high-assurance app store", December 2, 2013
44. University of Wisconsin Distinguished Lecture, "Verification games: Making software verification

fun", November 13, 2013
45. University of British Columbia ECE Distinguished Lecture, "Verification games: Making software verification fun", September 9, 2013
46. JavaOne, "Enhanced Metadata in Java SE 8", September 23, 2012
47. FuSE, "All work and no play makes software engineering dull" (keynote), July 17, 2013
48. Notkinfest, "Software evolution then and now: The research impact of David Notkin", February 1, 2013
49. JavaOne, "Build Your Own Type System for Fun and Profit", October 2, 2012
50. OSCON, "Preventing Runtime Errors at Compile Time", July 17, 2012
51. ICST, "Reproducible Tests? Non-duplicable Results in Testing and Verification" (keynote), April 20, 2012
52. Oracle, "Detecting and preventing bugs with pluggable type-checking", April 2012
53. Oracle, "JSR 308: Type Annotations: Making Java annotations more general and more useful", January 10, 2012
54. University of Maryland, "Always-available static and dynamic feedback: Unifying static and dynamic typing", November 8, 2011
55. University of Maryland, "Verification games: Crowd-sourced program verification", November 8, 2011
56. ICSE, "Always-available static and dynamic feedback", May 27, 2011
57. Oracle, "Detecting and preventing bugs with pluggable type-checking", March 2011
58. Usable Verification workshop, "User-specified type systems for domain-specific verification", Nov 16, 2010
59. Microsoft, RiSE all-hands meeting, August 5, 2010
60. TAP keynote, "How tests and proofs impede one another: The need for always-on static and dynamic feedback", July 1, 2010
61. Devoxx, "Detecting and preventing bugs with pluggable type-checking", November 2009
62. OOPSLA tutorial, "Preventing bugs with pluggable type checking", October 2009
63. University of Washington, "Preventing bugs with pluggable type checking", October 2009
64. OOPSLA tutorial, "Preventing bugs with pluggable type checking", October 26, 2009
65. SCAM keynote, "How analysis can hinder source code manipulation – and what to do about it", Sep 20, 2009
66. PPPJ tutorial, "Preventing bugs with pluggable type checking", Aug 27, 2009
67. IBM, "Self-defending software: Automatically patching security vulnerabilities", Aug 25, 2009
68. Microsoft, RiSE all-hands meeting, July 1, 2009
69. Microsoft, "Towards better software: Finding, fixing, and preventing errors", June 29, 2009
70. JavaOne, "Preventing bugs with pluggable type checking", June 2009
71. Seattle Java User's Group, "Preventing bugs with pluggable type checking", April 2009
72. Microsoft Research, RiSE Group, "Making it easier (and more fun!) to create software", April 21, 2009
73. Javoxx, "Preventing bugs with pluggable type-checking for Java", December 2008
74. Google, "Preventing bugs with pluggable type checking", December 2008
75. MPI-SWS Distinguished Lecture, "Self-defending software: Automatically patching security vulnerabilities", Nov 25, 2008
76. University of Kaiserslautern, "Self-defending software: Automatically patching security vulnerabilities", Nov 24, 2008
77. University of Karlsruhe, "Self-defending software: Automatically patching security vulnerabilities",

Nov 17, 2008
78. King's College London, CREST center, "Self-defending software: Automatically patching security vulnerabilities", Nov 12, 2008
79. OOPSLA, "Enforcing reference and object immutability in Java", October 2008
80. OOPSLA, "Compile-time type-checking for custom type qualifiers in Java", October 2008
81. ISSTA, "Practical pluggable types for Java", July 23, 2008
82. Max Planck Institute for Software Systems, "Practical pluggable types for Java", July 17, 2008
83. ECOOP, "ReCrash: Making software failures reproducible by preserving object states", July 11, 2008
84. ECOOP, "Inference of reference immutability", July 11, 2008
85. ECOOP, "Building and using pluggable type systems with the Checker Framework", July 10, 2008
86. University of Saarland, "ReCrash: Making software failures reproducible by preserving object states", July 8, 2008
87. University of Saarland, "Inference of reference immutability", July 4, 2008
88. JavaOne, "Upcoming Java programming-language changes", May 6, 2008 (with Alex Buckley)
89. Dagstuhl workshop Scalable Program Analysis, "Scalable pluggable types", April 14–18, 2008
90. J-Spring, "Preventing bugs with pluggable type-checking for Java", April 16, 2008
91. University of Washington, "Self-defending software: Collaborative learning for security", April 1, 2008
92. QCon, "User-defined type systems for error detection and prevention", November 9, 2007
93. OOPSLA, "Compile-time typechecking for custom Java type qualifiers", October 2007
94. Harvard University, "Refactoring for parameterizing Java classes", March 7, 2007
95. Stanford University, "Refactoring for parameterizing Java classes", January 8, 2007
96. Microsoft Research, "Dynamic inference of abstract types", December 12, 2006
97. University of California at Berkeley, "Combined static and dynamic mutability analysis", December 11, 2006
98. IBM T.J. Watson Research Center, "Combined static and dynamic mutability analysis", November 28, 2006
99. Microsoft Research India, "Evaluating systematic and random testing", October 10, 2006
100. Tata Research Development & Design Centre, "Combined static and dynamic mutability analysis", September 29, 2006
101. Indian Institute of Technology, Delhi, "Evaluating systematic and random testing", September 28, 2006
102. IBM India Research Lab, "Choosing the right tests: test selection via operational abstraction", September 28, 2006
103. Harvard University, "Javari: Adding reference immutability to Java", February 22, 2006
104. University of California at San Diego, "Javari: Adding reference immutability to Java", November 8, 2005
105. Tutorial at Conference on Automated Software Engineering 2005, "Learning from executions: Dynamic analysis for program understanding and software engineering", November 7, 2005
106. University of Arizona, "Eclat: Automatic generation and classification of test inputs", October 27, 2005
107. Conference on Verified Software: Theories, Tools, Experiments, "Verification for legacy programs", October 10, 2005
108. DARPA Self-Regenerative Systems meeting, "Learning and repair techniques for self-healing systems", July 12, 2005
109. ABB, "Making the most of impoverished test suites: Automatic classification of test inputs and test

factoring", November 19, 2004

110. OOPSLA Workshop on the Java Platform: Tiger and Beyond, "Integrating reference immutability into the Java mainstream", October 28, 2004
111. Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), "A practical type system and language for reference immutability", October 26, 2004
112. Philips Medical, "Making the most of impoverished test suites: Automatic classification of test inputs and test factoring", October 4, 2004
113. UW/MSR Summer Institute on Trends in Testing: Theory, Techniques and Tools (keynote), "Making the most of impoverished test suites: Automatic classification of test inputs and test factoring", August 24, 2004
114. Program Analysis and Software Tools for Engineering (PASTE) workshop (keynote), "Static and dynamic analysis: Synergy and duality", June 7, 2004
115. Carnegie Mellon University, "Improving adaptability via program steering", April 27, 2004
116. University of Limerick, "Using dynamic analysis to assist theorem proving", December 12, 2003
117. Cambridge University and Microsoft Research, "Predicting problems caused by component upgrades", December 10, 2003
118. Dagstuhl workshop Understanding Program Dynamics, "Comparing dynamic program behaviors", December 2, 2003
119. Dagstuhl workshop Understanding Program Dynamics (keynote), "Static and dynamic analysis: Synergy and duality", December 1, 2003
120. IBM T.J. Watson Research Center (Distinguished Lecture), "Improving test suites via operational abstraction", June 16, 2003
121. ICSE Workshop on Dynamic Analysis (WODA), Portland, Oregon, "Static and dynamic analysis: Synergy and duality", May 9, 2003
122. International Conference on Software Engineering (ICSE), Portland, Oregon, "Improving test suites via operational abstraction", May 6, 2003
123. Microsoft Research, Portland, Oregon, "Improving test suites via operational abstraction", May 2, 2003
124. University of Pittsburgh, "Predicting problems caused by component upgrades", February 4, 2003
125. Carnegie Mellon University, "Improving test suites via operational abstraction", February 3, 2003
126. Cambridge University, England, "Using dynamic analysis to assist program verification", July 25, 2002
127. Java Verification Workshop (JVW'01), Portland, Oregon, "Static verification of dynamically detected program invariants", January 13, 2002
128. McMaster University, "Refactoring and static verification: Two applications of dynamic invariant detection", January 25, 2002
129. Rice University, "Refactoring and static verification: Two applications of dynamic invariant detection", January 15, 2002
130. University of Washington, "Refactoring and static verification: Two applications of dynamic invariant detection", January 10, 2002
131. Williams College, "Refactoring and static verification: Two applications of dynamic invariant detection", November 30, 2001
132. International Conference on Software Maintenance (ICSM), Florence, Italy, "Automated Support for Program Refactoring using Invariants", November 9, 2001
133. International Conference on Software Maintenance (ICSM), Florence, Italy, "Dynamically Detecting Likely Program Invariants", November 8, 2001
134. IBM T.J. Watson Research Center, "Refactoring and static verification: Two applications of dynamic

invariant detection", August 8, 2001
135. Brunel University, "Overview of dynamic invariant detection", July 24, 2001
136. Workshop on Runtime Verification (RV'01), Paris, France, "Static verification of dynamically detected program invariants", July 23, 2001
137. International Conference on Software Engineering (ICSE), Montreal, Canada, "The future of software engineering" (panel), May 18, 2001
138. University of Wisconsin at Madison, "Refactoring and static verification: Two applications of dynamic invariant detection", May 14, 2001
139. University of California at Irvine, "Dynamically Detecting Likely Program Invariants", February 23, 2001
140. Compaq Systems Research Center, "Dynamically Detecting Likely Program Invariants", February 22, 2001
141. University of Southern California, "Dynamically Detecting Likely Program Invariants", February 21, 2001
142. MIT, "Playing Konane Mathematically with Combinatorial Game Theory", January 17, 2001
143. Brown University, "Dynamically Detecting Likely Program Invariants", February 1, 2001
144. Tufts University, "Dynamically Detecting Likely Program Invariants", November 20, 2000
145. University of Virginia, "Top Gun" Distinguished Lecture Series, "Dynamically Detecting Likely Program Invariants", November 13, 2000
146. Boston University, "Dynamically Detecting Likely Program Invariants", September 25, 2000
147. Massachusetts Institute of Technology, "Dynamically Detecting Likely Program Invariants", April 10, 2000
148. University of Maryland, "Dynamically Detecting Likely Program Invariants", April 5, 2000
149. Carnegie Mellon University, "Dynamically Detecting Likely Program Invariants", April 3, 2000
150. University of Massachusetts at Amherst, "Dynamically Detecting Likely Program Invariants", March 29, 2000
151. Georgia Institute of Technology, "Dynamically Detecting Likely Program Invariants", March 23, 2000
152. Stanford University, "Dynamically Detecting Likely Program Invariants", March 6, 2000
153. University of California at Berkeley, "Dynamically Detecting Likely Program Invariants", March 2, 2000
154. University of California at San Diego, "Dynamically Detecting Likely Program Invariants", February 29, 2000
155. University of Colorado at Boulder, "Dynamically Detecting Likely Program Invariants", February 24, 2000
156. Cornell University, "Dynamically Detecting Likely Program Invariants", February 15, 2000

Presentations given at conferences

All of the conference proceedings articles listed earlier (page 5), plus others that were superseded by a later publication and so do not appear on this cv, were presented at a conference. Most were given by my coauthors, such as students, and I do not list those here.

Professional society memberships

Association for Computing Machinery (ACM) since 1996
Eta Kappa Nu (electrical engineering honor society) since 1987
Institute for Electical and Electronics Engineering (IEEE) since 2013
Sigma Xi (scientific research honor society) since 1987
Tau Beta Pi (engineering honor society) since 1987

Program committees and journal editorships

1. BenchWork 2019.
2. SecDev 2017. Conference on Secure Development.
3. FSE 2016 SRC. Student Research Competition Committee, ACM SIGSOFT International Symposium on the Foundations of Software Engineering.
4. SecDev 2016. Conference on Secure Development.
5. PLDI 2016 EPC. PLDI 2016 DPC. ACM Conference on Programming Language Design and Implementation, External Program Committee and Distinguished Paper Committee.
6. OOPSLA 2016 ERC. Conference on Object-Oriented Programming Systems, Languages and Applications, External Review Committee.
7. ICSE 2016. 38th International Conference on Software Engineering.
8. ESEC/FSE 2015 DS. Doctoral Symposium of the 11th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering.
9. GAS 2015. 4th International Workshop on Games and Software Engineering.
10. PLDI 2014 ERC. ACM Conference on Programming Language Design and Implementation, External Review Committee.
11. ACM SIGSOFT Dissertation Award 2014.
12. ESEC/FSE 2013. 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering.
13. Co-chair, FuSE 2013. Future of Software Engineering symposium.
14. Co-chair, ICSE 2013 tutorials. 35th International Conference on Software Engineering.
15. ICSE 2013 Mentoring Committee. 35th International Conference on Software Engineering.
16. WODA 2013. 11th International Workshop on Dynamic Analysis.
17. Chair, SPLASH 2012 Panels. Systems, Programming, Languages and Applications: Software for Humanity.
18. SPLASH 2012 Demos. Systems, Programming, Languages and Applications: Software for Humanity.
19. SPLASH 2012 Doctoral symposium. Systems, Programming, Languages and Applications: Software for Humanity.
20. OOPSLA 2011. Conference on Object-Oriented Programming Systems, Languages and Applications
21. ICSE 2011. 33rd International Conference on Software Engineering
22. TAIC PART 2010. Testing: Academic and Industrial Conference - Practice and Research Techniques
23. WODA 2009. Workshop on Dynamic Analysis,
24. ICSE 2009 Mentor Program. 31st International Conference on Software Engineering
25. WODA 2009. Workshop on Dynamic Analysis
26. ICSE 2009 Mentor Program. 31st International Conference on Software Engineering
27. HotSWUp 2008. ACM Workshop on Hot Topics in Software Upgrades

28. ISSTA 2008. International Symposium on Software Testing and Analysis
29. ESEC-FSE 2007. European Software Engineering Conference and ACM International Symposium on the Foundations of Software Engineering
30. TAP 2007. Tests and Proofs
31. ICSE 2007 Education Track. 29th International Conference on Software Engineering
32. eTX 2006. Eclipse Technology Exchange Workshop
33. ECOOP 2006. European Conference on Object-Oriented Programming
34. ISSTA 2006. International Symposium on Software Testing and Analysis
35. PLDI 2006. ACM Conference on Programming Language Design and Implementation
36. WODA 2006. Workshop on Dynamic Analysis
37. CC 2006. International Conference on Compiler Construction
38. VSTTE 2005. Verified Software: Theories, Tools, Experiments
39. Co-chair, PASTE 2005. ACM Workshop on Program Analysis for Software Tools and Engineering
40. ESDDT 2005 ("Bugs 2005"). Workshop on the Evaluation of Software Defect Detection Tools
41. POPL 2005. ACM Symposium on Principles of Programming Languages
42. FTfJP 2004. Formal Techniques for Java-like Programs Workshop
43. PASTE 2004. ACM Workshop on Program Analysis for Software Tools and Engineering
44. WODA 2004. Workshop on Dynamic Analysis
45. ICSE 2004. 26th International Conference on Software Engineering
46. PL Day 2004. IBM Programming Language Day Workshop
47. eTX 2004. Eclipse Technology Exchange Workshop
48. Co-chair, WODA 2003. Workshop on Dynamic Analysis
49. PASTE 2002. ACM Workshop on Program Analysis for Software Tools and Engineering
50. FSE 2002. ACM Tenth International Symposium on the Foundations of Software Engineering
51. RV 2002. Second Workshop on Runtime Verification
52. NEPLS 6. Sixth New England Programming Languages and Systems Symposium, 2002
53. Chair, NEPLS 5. Fifth New England Programming Languages and Systems Symposium, 2002
54. SCAM 2001. IEEE International Workshop on Source Code Analysis and Manipulation
55. NEPLS 4. Fourth New England Programming Languages and Systems Symposium, 2001
56. RV 2001. Workshop on Runtime Verification
57. Chair, IR 1995. ACM SIGPLAN Intermediate Representations Workshop

In addition, I have been an external reviewer for too many additional conferences to count, and likewise have done many journal reviews. Here is a rather incomplete list, as of 2006: Computer-aided Verification (CAV), Empirical Software Engineering Journal, Foundations of Software Engineering (FSE), International Conference on Software Engineering (ICSE) education track, International Joint Conference on Artificial Intelligence (IJCAI), International Journal of Software and Informatics (IJSI), International Symposium on Software Testing and Analysis (ISSTA), ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), ACM Conference on Programming Language Design and Implementation (PLDI), ACM Symposium on Principles of Programming Languages (POPL), ACM European SigOps Workshop, ACM Symposium on Operating System Principles (SOSP), International Workshop on Test and Analysis of Component Based Systems (TACoS), ACM Transactions on Programming Languages and Systems (TOPLAS), ACM Transactions on Software Engineering and Methodology (TOSEM), IEEE Transactions on Dependable and Secure Computing (TDSC), IEEE Transactions on Software Engineering (TSE).